NORMALIZZAZIONE DI SCHEMI RELAZIONALI

- Lo scopo della normalizzazione e' di progettare dei "buoni" schemi di basi di dati; gli schemi normalizzati evitano specifiche *anomalie*

- La teoria della normalizzazione e' stata sviluppata per il modello relazionale

- E' da notare che si usano metodologie basate su una definizione in termini del modello E-R dello schema e successiva traduzione di questo in termini del modello relazionale gran parte delle anomalie non si verificano

- Le anomalie che si vuole evitare sono le seguenti:

  1) ripetizione di informazioni (ridondanza)
  2) potenziali inconsistenze nelle modifiche
  3) problemi nelle inserzioni e nelle cancellazioni

ANOMALIE

Ridondanza

- Consideriamo la seguente relazione

  supplier (sname, saddress, item, price)

  con chiave (*sname, item*)
  e supponiamo che un fornitore abbia un solo indirizzo e possa fornire piu' prodotti

- Un problema di questo schema e' che l'indirizzo del fornitore e' ripetuto per ogni prodotto fornito dal fornitore

Esempio:

supplier

sname	*saddress*	*item*	*price*
ZackZebra	FamilyWay	Brie	3.49
ZackZebra	FamilyWay	Perrier	1.19
JudyGiraffe	RockyRoad	Macadamias	0.06
RuthRino	LoisLane	Perrier	3.98
RuthRino	LoisLane	Escargot	7.00

ANOMALIE

Inconsistenze nelle modifiche

- A causa della ridondanza si possono verificare anomalie quando si eseguono modifiche

- Supponiamo di voler modificare l'indirizzo del fornitore di nome 'ZackZebra' da 'FamilyWay' a 'ViaAlberti':

  se non si ha la precauzione si modificarlo in tutte le tuple in cui 'ZackZebra' appare, si possono avere tuple con indirizzi diversi per lo stesso fornitore

ANOMALIE

Problemi di inserzione

- Non possiamo inserire un fornitore se non fornisce almeno un prodotto (nota che la chiave e' fornita sia dal nome del fornitore e dal nome del prodotto e non e' possibile inserire tuple con chiavi parzialmente nulle)

Problemi di cancellazione

- Se cancelliamo tutti i prodotti forniti da un fornitore, perdiamo traccia del fornitore

Notare che tutti i problemi precedenti non si verificano se usiamo uno schema del tipo:

sa (*sname, saddress*)	con chiave *sname*
sip(*sname, price, item)*	con chiave (*sname, item*)

NORMALIZZAZIONE

- Il processo di normalizzazione consiste nel determinare degli schemi di relazioni che non hanno i problemi visti

- Molto spesso la normalizzazione conduce a definire relazioni con meno attributi

- E' da notare che la normalizzazione puo' portare a delle inefficienze nelle prestazioni:

  per esempio nel secondo schema dobbiamo
  eseguire un join per determinare l'indirizzo
  del fornitore di un prodotto

  questo non e' necessario nel primo schema

- La normalizzazione e' basata sul concetto delle *Dipendenze Funzionali*

DIPENDENZE FUNZIONALI

- Sia R(A1,A2,....An) uno schema di relazione, e siano X e Y sottoinsiemi di {A1,A2,....An}:

  si dice che "X determina funzionalmente Y"
  e si indica con X->Y se per
  qualunque *r* stato (o estensione) di R non
  e' possibile che due tuple in *r*
  che hanno valori uguali per gli attributi in X, abbiano valori diversi per gli attributi in Y

- In altre parole:

  se t1, t2 c *r* e t1.X=t2.X, allora t1.Y=t2.Y

- Le dipendenze funzionali esprimono vincoli di integrita' sulla base dei dati

- Alcune abbreviazioni utili:

  si usa A1 A2 .....An come abbreviazione di
  {A1, A2,.....,An}

  dati X e Y insiemi di attributi XY denota X U Y

DIPENDENZE FUNZIONALI

Esempio

Consideriamo un database con il seguente schema:

Members (Name, Address, Balance)
Orders (Order_no, Name, Item, Quantity)
Suppliers(Sname, Saddress, Item Price)

Supponiamo che ogni cliente abbia un unico indirizzo e un unico bilancio e che ogni cliente abbia nome diverso. Queste assunzioni sono formalizzate dalla seguente dipendenza funzionale

Name -> Address Balance

Nella relazione Orders possiamo assumere che il numero di ordine determini tutto, poiche' ordini diversi hanno numero di ordine diverso, quindi:

Order_no -> Name Item Quantity

Nella relazione Suppliers osserviamo le seguenti dipendenze:
Sname -> Saddress
Sname Item -> Price

DIPENDENZE FUNZIONALI

Esempio (continua)

- Una domanda e' se dipendenze quali ad esempio:

  Saddress -> Sname o Address -> Name

  siano valide.

- La risposta dipende dalla semantica dei dati: se noi ammettiamo che due diversi clienti possano avere lo stesso indirizzo, allora la dipendenza

  Address -> Name   non e' valida

  (infatti dato uno stesso indirizzo, posso trovare due diversi clienti con nome diverso)

- Inoltre valgono delle dipendenze banali come ad esempio:

  1) Name -> Name

  2) Sname Item -> Item

  Queste dipendenze affermano:

  1) che due tuple che hanno lo stesso valore per Name hanno lo stesso valore per Name
  2) che due tuple che hanno lo stesso valore per Item e Sname hanno lo stesso valore per Item

DIPENDENZE FUNZIONALI

Esempio (continua)

- Altre dipendenze non banali sono derivate usando le dipendenze che abbiamo gia' stabilito

- Esempio:

  Sname Item -> Saddress Price

- Questa dipendenza dice che due tuple che hanno lo stesso valore per Sname e Item devono avere lo stesso valore per entrambi gli attributi Saddress e Price

- Questa dipendenza viene ottenuta osservando che Sname da solo determina Saddress, mentre Sname e Item determinano Price

- Quindi fissando un valore di Sname e uno di Item troviamo un solo valore per Saddress e uno solo per Price

DIPENDENZE FUNZIONALI

Implicazioni logiche

- Supponiamo che:
  R sia uno schema di relazione
  A, B, e C alcuni degli attributi di R
  A -> B e B -> C
  allora: A -> C (*)

  (nel seguito si usa la notazione compatta
  A -> B C per indicare
  A ->B e A -> C)

- Dato un insieme *F* di dipendenze funzionali per uno schema R, e una dipendenza funzionale X ->Y, si dice che *F* *implica logicamente* X -> Y, scritto *F* |= X -> Y, se ogni relazione *r* di schema R che verifica le dipendenze funzionali in *F*, verifica anche X -> Y

Esempio: {A -> B , B -> C} |= A -> C

- La *chiusura* di un insieme *F* di dipendenze funzionali e' definita come l'insieme delle dipendenze funzionali implicate da *F*
  F+ = {X -> Y / *F* |= X -> Y}

DIPENDENZE FUNZIONALI

Implicazioni logiche

Esempio: R(A,B,C) e *F* = {A->B, B->C}
F+ consiste di tutte le dipendenze funzionali X->Y che verificano una delle tre condizioni

1. X contiene A, cioe' X->Y e'
   ABC -> BC, AB -> BC, oppure A -> C

2. X contiene B ma non A e Y non contiene A, cioe'
   X->Y e' BC -> B

3. X -> Y e' la dipendenza C -> C

Esiste un algoritmo che determina le dipendenze funzionali

DIPENDENZE FUNZIONALI

Chiavi

- Abbiamo gia' introdotto il concetto di chiave

- Esiste un concetto analogo nel caso delle dipendenze funzionali

- Dato uno schema R con attributi A1 A2.....An e un insieme di dipendenze funzionali *F* e X sottoinsieme di R, si dice che X e' una *chiave* di R se:

  1. X -> A1 A2.....An e' in *F*+
     cioe' la dipendenza di tutti gli attributi da X o e' data o segue logicamente dalle dipendenze funzionali date

  2. non esiste Y sottoinsieme proprio di X tale che
     Y -> A1 A2.....An e' in *F*+

  (La seconda condizione implica una condizione di minimalita')

- Si usa il termine *superchiave* per indicare un soprainsieme della chiave

DIPENDENZE FUNZIONALI

Chiavi

Esempio 1

Persona(CodiceFiscale, Nome, Cognome, DataNascita)

F={CodiceFiscale -> Nome Cognome DataNascita}

(questa dipendenza dice che il codice fiscale e' unico per ogni persona, cioe' non esistono due o piu' tuple che hanno uno stesso valore di CodiceFiscale e diversi valori per gli altri attributi)

E' facile vedere che CodiceFiscale e' una chiave per Persona, mentre
CodiceFiscale Cognome non lo e'
in quanto ha un sottoinsieme proprio (CodiceFiscale) che determina funzionalmente tutti gli attributi

Esempio 2

R(A,B,C) e *F* = {A->B, B->C}
La chiave e' A
infatti A->A, A->B, A->C (per la proprieta' transitiva)
o equivalentemente A->ABC
inoltre non esiste un sottoinsieme degli attributi di R non contenente A tale che X->ABC

DIPENDENZE FUNZIONALI

Assiomi per le dipendenze funzionali

- Per determinare quale insieme di attributi di una relazione costituisce una chiave, dato un insieme di dipendenze funzionali *F*, dobbiamo calcolare l'insieme *F+* da *F* , o almeno
  data la dipendenza funzionale X -> Y,
  determinare se X -> Y e' in *F+*

- E' stato definito un insieme di regole di inferenza che permettono di calcolare tutte le dipendenze in *F+*

- Tali regole sono *sound* nel senso che non si possono dedurre regole che non sono in *F+*

- Questo insieme di regole e' chiamato
  assiomi di Armstrong
  (dal nome della persona che le ha definite nel 1974)

DIPENDENZE FUNZIONALI

Assiomi di Armstrong

Dato un insieme di attributi D e un insieme *F* di dipendenze funzionali definite per gli attributi in D, le regole di inferenza sono le seguenti:

A1: (*riflessivita'*). Se $Y \subseteq X \subseteq D$, allora X -> Y e' logicamente implicata da *F*.

Questa regola genera *dipendenze banali*, cioe' dipendenze in cui la parte destra e' contenuta nella parte sinistra; l'uso di questa regola non dipende da *F*

esempio: $A \subseteq AB$, quindi AB -> A

A2: (*additivita'*). Se X -> Y, e $Z \subseteq D$, allora XZ -> YZ e' implicata da *F*.

Notare che se X, Y, e Z sono insiemi di attributi , XZ e' un'abbreviazione per X U Z;

notare inoltre che che X -> Y potrebbe essere direttamente presente in *F* o essere derivata usando le regole che stiamo vedendo

A3: (*transitivita'*). Se X -> Y e Y -> Z, allora X -> Z e' implicata da *F*.

DIPENDENZE FUNZIONALI

Assiomi di Armstrong

Teorema 1 : Gli assiomi di Armstrong sono sound e completi.

E' possibile derivare altre regole dagli assioni di Armstrong:

a) Regola di unione

$\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$

b) Regola di pseudotransitivita'

$\{X \rightarrow Y, WY \rightarrow Z\} \models XW \rightarrow Z$

c) Regola di decomposizione

Se $X \rightarrow Y$ vale, e $Z \subseteq Y$, allora $X \rightarrow Z$ vale

Una conseguenza importante della regola di unione e' che se A1,....,An sono attributi, allora

$X \rightarrow A1,......, An$ vale se e solo se $X \rightarrow Ai$ per ogni i=1,...,n

DIPENDENZE FUNZIONALI

Chiusura di un insieme di attributi rispetto ad un insieme di dipendenze funzionali

Dato *F* insieme di dipendenze funzionali su un insieme di attributi D, e X sottoinsieme di D
la *chiusura* (closure) di X rispetto a *F* (denotata da X+) e' definita come un insieme di attributi

X+ = {A / X->A e' derivabile da *F* mediante gli assiomi di Armstrong}

Per la soundness e completezza degli assiomi di Armstrong si ha equivalentemente

X+ = {A /*F* implica logicamente X -> A}

DIPENDENZE FUNZIONALI

Calcolo della chiusura transitiva

- Consideriamo l'insieme *F* ={A->B1, A->B2,.......,A->Bn}

  F+ include tutte le dipendenze A->Y con Y sottoinsieme di {B1,B2,....Bn}.
  Ci sono 2**n sottoinsiemi Y, quindi il calcolo di *F*+ e' piuttosto costoso

- Viceversa calcolare la chiusura transitiva rispetto ad un insieme di attributi X e' meno costoso;
  inoltre sempre per il Lemma 3, verificare se X->Y e' in *F*+ non e' piu' complesso del calcolo di X+

Notare che a noi non interessa calcolare tutte le dipendenze funzionali logicamente implicate da un dato insieme *F*

Piuttosto interessa verificare se un certo insieme di attributi X implica tutti gli altri in modo da poter determinare se X e' una chiave per la relazione

DIPENDENZE FUNZIONALI

Calcolo della chiusura transitiva

- *Algoritmo 1* Calcolo della chiusura di un insieme di attributi rispetto ad un insieme di dipendenze funzionali

Input Un insieme finito D di attributi, un insieme di dipendenze funzionali *F*, e un insieme X $\subseteq$ D

Output X+, chiusura di X rispetto ad *F*

Approccio Si calcola una sequenza di insiemi di attributi X(0), X(1),....... usando le seguenti regole

1. X(0) e' X
2. X(i+1) e' X(i) piu' un insieme di attributi A tali che:
   - c'e' una dipendenza Y->Z in *F*
   - A e' in Z
   - Y $\subseteq$ X(i)

Poiche' X=X(0) $\subseteq$ ......$\subseteq$ X(i) $\subseteq$.........$\subseteq$ D e D e' finito, alla fine si raggiunge un i tale che
X(i)=X(i+1);
ne segue che X(i)=X(i+1)=X(i+2)=.......
Quindi l'algoritmo si ferma non appena si determina che X(i)=X(i+1)

DIPENDENZE FUNZIONALI

Calcolo della chiusura transitiva

- Si dimostra che X+ e' X(i) per un i tale che
  X(i)=X(i+1)

Esempio: Sia *F* il seguente insieme

AB->C	C->A	BC->D	ACD->B
D->EG	BE->C	CG->BD	CE->AG

Sia X=BD

X(0)=BD

X(1): dobbiamo determinare tutte le dip.funzionali aventi sul lato sinistro B, D, o BD
Ne esiste solo una D->EG
X(1)=BDEG

X(2): dobbiamo determinare tutte le dip. funzionali aventi sul lato sinistro qualche attributo in X(1)
Ne esistono due: D->EG e BE->C
X(2)=BCDEG

X(3): determiniamo le seguenti dip. funzionali (in aggiunta a quelle trovate nei passi precedenti)
C->A, BC->D, CG->BD, e CE->AG
Quindi X(3)=ABCDEG (BD)+=X(3)

DIPENDENZE FUNZIONALI
Insiemi di copertura

- Siano *F* e *G* insiemi di dipendenze funzionali, si dice che *F* e *G* sono *equivalenti* se *F*+=*G*+ (*F* "copre" *G* e viceversa)

- Per determinare l'equivalenza si procede come segue: per ogni dipendenza Y->Z in *F*, si testa se Y->Z e' in *G* + usando l'algoritmo 1 per calcolare Y+ e quindi si vede se $Z \subseteq$ Y+

  se una qualche dipendenza Y->Z in *F* non e' in *G*+, sicuramente *F*+~=*G*+

  Si esegue il viceversa determinando se ogni dipendenza funzionale in *G* e' in *F*+

Ogni insieme di dipendenze funzionali *F* e' coperto da un insieme di dipendenze *G* in cui nessun lato destro ha piu' di un attributo

DIPENDENZE FUNZIONALI

Insiemi minimali

Un insieme di dipendenze *F* e' *minimo* se:

1. Il lato destro di ogni dipendenza in *F* e' un singolo attributo
2. Per nessuna X->A in *F* l'insieme *F* - {X->A} e' equivalente a *F*
3. Per nessuna X->A in *F* e nessun sottoinsieme Z di X l'insieme *F* - {X->A} U {Z->A} e' equivalente a *F*

Intuitivamente, la condizione (2) garantisce che in *F* non ci siano dipendenze ridondanti;
la condizione (3) garantisce che sul lato sinistro non ci siano attributi ridondanti.
Poiche' sul lato destro c'e' un solo attributo in base alla condizione (1), sicuramente non c'e' nessun lato destro ridondante.

Teorema 3 : Ogni insieme di dipendenze *F* e' equivalente ad un insieme minimale *F'*.

DIPENDENZE FUNZIONALI

Insiemi minimali

Per determinare l'insieme minimale si procede come segue:

1) Si genera F' da F sostituendo ad ogni dipendenza X->Y dove dove Y ha la forma A1....An un insieme di dipendenze di forma X->A1, ..., X->An

2) Si considera ogni dipendenza in F' che abbia piu' di un attributo sul lato sinistro. Se e' possibile eliminare un attributo dal lato sinistro e avere ancora un insieme di dipendenze equivalente, si elimina tale attributo. Sia F'' l'insieme generato.
(approccio: B in X e' ridondante rispetto alla dipendenza X->A
se (X - {B})+ calcolata rispetto a F' contiene A)

3) Si considera ogni dipendenza X->A in F'' in un qualche ordine e se F''- {X->A} e' equivalente a F'', si cancella X->A da F''
(approccio: si calcola X+ rispetto a F''- {X->A}
se X+ include A, allora X->A e' ridondante)

DIPENDENZE FUNZIONALI

Insiemi minimali

Notare che al passo (2) l'ordine in base a cui le dipendenze funzionali sono esaminate al passo (2) puo' generare insiemi minimali diversi:

A->B	A->C
B->A	C->A
B->C	

e' possibile eliminare B->A e A->C,
oppure B->C ma non tutte e tre

Notare che al passo (3) l'ordine in cui gli attributi sono esaminati influenza il risultato che si ottiene

AB->C   A->B   B->A

Possiamo eliminare o A o B da AB->C ma non entrambi

Insiemi minimali - esempio

Sia *F* il seguente insieme

AB->C C->A BC->D ACD->B
D->EG BE->C CG->BD CE->AG

Applicando il Lemma 4 per semplificare i lati destri otteniamo

AB->C C->A BC->D ACD->B
D->E D->G BE->C CG->B
CG->D CE->A CE->G

Si nota che:

- CE->A e' ridondante in quanto e' implicato da C->A
- CG->B e' ridondante in quanto
  CG->D, C->A, e ACD->B implicano CG->B come puo' essere controllato calcolando (CG)+
- Inoltre ACD->B puo' essere sostituito da CD->B poiche' C->A e' dato

Risultato:

AB->C C->A BC->D CD->B D->E D->G
BE->C CG->D CE->G

DIPENDENZE FUNZIONALI

Insiemi minimali - esempio

E' possibile ottenere un altro insieme di copertura minimale eliminando:

CE->A, CG->D, e ACD->B

L'insieme ottenuto e':

AB->C	C->A	BC->D	D->E
D->G	BE->C	CG->B	CE->G

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

- La scomposizione di uno schema relazionale R={A1, A2,.......,An} e' la sua sostituzione con una collezione C={R1, R2, ...,Rk} di sottoinsiemi di R tali che R = R1 U R2 U......Rk

- Non si richiede che i vari schemi Ri siano disgiunti

- Uno dei motivi per eseguire una scomposizione di uno schema relazionale e' che si eliminano alcune delle anomalie viste

Esempio:
supplier (*sname, saddress, item, price)*
con dipendenze funzionali

sname ->*saddress* e
sname item -> *price*

una scomposizione possibile di supplier e'

SA (*sname, saddress)* e *SIP(sname , item, price)*

questa scomposizione elimina le anomalie viste: quindi il problema della scomposizione ottimale e' legato alle dipendenze funzionali

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Lossless joins

- E' necessario fare attenzione a non perdere informazioni quando si esegue la scomposizione di uno schema di relazione in piu' schemi

- Dato uno schema di relazione R, una scomposizione R1,R2,.....Rk di R, e un insieme di dipendenze funzionali D, si dice che la scomposizione e' *lossless join* (rispetto a D) se per ogni relazione r di schema R che soddisfa D si ha:

  r =

  cioe' r e' il join naturale delle sue proiezioni sui vari schemi Ri (i=1,..,k)

- In altre parole, se una relazione viene scomposta, e' importante che sia possibile riottenere esattamente la stessa eseguendo il join naturale delle relazioni in cui e' stata scomposta

- E' importante che una scomposizione abbia la proprieta' di lossless join

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Lossless joins

Esempio:

Dato lo schema di relazione R(ABCDEGHI) e l'insieme

F = {A->HI, B->DE}

tutte le relazioni che soddisfano *F* sono scomponibili senza perdita rispetto ad ABDEI e ABCGH

Invece non sono scomponibili senza perdita rispetto ad ABDEI e ACGH

r	A	B	C	D	E	G	H	I
	a1	b1	c1	d1	e1	g1	h1	i1
	a1	b2	c2	d2	e2	g2	h1	i1

A	B	D	E	I
a1	b1	d1	e1	i1
a1	b2	d2	e2	i1

A	C	G	H
a1	c1	g1	h1
a1	c2	g2	h1

A	B	C	D	E	G	H	I
a1	b1	c1	d1	e1	g1	h1	i1
a1	b2	c2	d2	e2	g2	h1	i1
a1	b1	c2	d1	e1	g2	h1	i1
a1	b2	c1	d2	e2	g1	h1	i1

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Lossless joins

- Notazione addizionale:
  data una scomposizione ρ = (R1,R2,.....,Rk),

  $m_\rho (r) = \quad \pi_{Ri}(r)$

  denota il join naturale delle proiezioni di r sugli schemi di relazione in ρ

- Pertanto la condizione di lossless join e' espressa come segue:
  dato un insieme di dipendenze funzionali *F*, per ogni r che verifica *F*, deve valere $r=m_\rho (r)$

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Determinazione di scomposizioni lossless joins

Algoritmo 2:

Input: Uno schema di relazione R=A1....An, un insieme di dipendenze funzionali *F*, e una scomposizione ρ = (R1,R2,.....,Rk)

Output: Verifica se la scomposizione ha la proprieta' di lossless join

Metodo:

- Si costruisce una tabella con *n* colonne e *k* righe; la colonna *j* corrisponde all'attributo Aj, e la riga *i* corrisponde allo schema di relazione Ri
- Si assegna all'elemento (i,j) il simbolo a(j) se Aj e' in Ri; altrimenti di assegna b(i,j)
- Si considera ogni dipendenza funzionale X->Y in *F* fino a che la tabella non viene piu' modificata:

  ogni volta che si esamina X->Y, si considerano le righe che hanno lo stesso valore in tutte le colonne relative agli attributi in X: se due di tali righe esistono, i simboli per gli attributi in Y in tali righe vengono unificati come segue:

  se uno dei due simboli e' a(j), si pone l'altro ad a(j);

  se i due simboli sono rispettivamente b(i,j) e b(l,j) si unificano scegliendo il simbolo con il primo indice minore
- Se dopo aver modificato la tabella, si scopre che una qualche riga ha la forma a1,.....,ak allora la scomposizione e' lossless join, altrimenti e' *lossy*

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Determinazione di scomposizioni lossless joins

Esempio (1) : supplier (sname, saddress, item, price)
e dipendenze funzionali
sname->saddress   sname item->price
Verifichiamo se
SA(sname, saddress) e SIP(sname, item, price)
e' una scomposizione lossless join

	sname	saddress	item	price
SA	a(1)	a(2)	b(1,3)	b(1,4)
SIP	a(1)	b(2,2)	a(3)	a(4)

Si considera la dipendenza funzionale
sname->saddress
poiche' esistono due righe che hanno lo stesso valore per l'attributo sname (parte X della dipendenza), i valori dell'attributo saddress (parte Y della dipendenza) vengono unificati:

a(2) e b(2,2) vengono unificati ad a(2)

Pertanto otteniamo che

	sname	saddress	item	price
SA	a(1)	a(2)	b(1,3)	b(1,4)
SIP	a(1)	a(2)	a(3)	a(4)

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Determinazione di scomposizioni lossless joins

Esempio (2) :

- sia R=ABCDE uno schema di relazione
- sia R1=AD R2=AB R3=BE R4=CDE R5=AE una scomposizione di R
- siano date le seguenti dipendenze funzionali

A->C DE->C

B->C CE->A C->D

determinare se la scomposizione ha la proprieta' di lossless join

Matrice iniziale

	A	B	C	D	E
R1	a(1)	b(1,2)	b(1,3)	a(4)	b(1,5)
R2	a(1)	a(2)	b(2,3)	b(2,4)	b(2,5)
R3	b(3,1)	a(2)	b(3,3)	b(3,4)	a(5)
R4	b(4,1)	b(4,2)	a(3)	a(4)	a(5)
R5	a(1)	b(5,2)	b(5,3)	b(5,4)	a(5)

Applichiamo A->C, poiche' esistono tre righe (R1,R2,R5) che hanno lo stesso valore per A, possiamo unificare i simboli nella colonna di C per queste tre righe: b(1,3) b(2,3) b(5,3) (b(1,3) e' il simbolo rappresentante)

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Determinazione di scomposizioni lossless joins

Esempio (2) (continua)

Otteniamo la seguente matrice

	A	B	C	D	E
R1	a(1)	b(1,2)	b(1,3)	a(4)	b(1,5)
R2	a(1)	a(2)	b(1,3)	b(2,4)	b(2,5)
R3	b(3,1)	a(2)	b(3,3)	b(3,4)	a(5)
R4	b(4,1)	b(4,2)	a(3)	a(4)	a(5)
R5	a(1)	b(5,2)	b(1,3)	b(5,4)	a(5)

Consideriamo la dipendenza B->C che ci permette di unificare i simboli b(1,3) e b(3,3) per le righe R2 e R3 (b(1,3) e' il simbolo rappresentante)

Otteniamo la seguente matrice

	A	B	C	D	E
R1	a(1)	b(1,2)	b(1,3)	a(4)	b(1,5)
R2	a(1)	a(2)	b(1,3)	b(2,4)	b(2,5)
R3	b(3,1)	a(2)	b(1,3)	b(3,4)	a(5)
R4	b(4,1)	b(4,2)	a(3)	a(4)	a(5)
R5	a(1)	b(5,2)	b(1,3)	b(5,4)	a(5)

Consideriamo la dipendenza C->D

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Determinazione di scomposizioni lossless joins

Esempio (2) (continua)

La dipendenza C->D ci permette di unificare i simboli nella colonna D
a(4), b(2,4), b(3,4), b(5,4) per le righe R1,R2,R3,R5
Il simbolo scelto come rappresentante e' a(4)

Otteniamo la seguente matrice

	A	B	C	D	E
R1	a(1)	b(1,2)	b(1,3)	a(4)	b(1,5)
R2	a(1)	a(2)	b(1,3)	a(4)	b(2,5)
R3	b(3,1)	a(2)	b(1,3)	a(4)	a(5)
R4	b(4,1)	b(4,2)	a(3)	a(4)	a(5)
R5	a(1)	b(5,2)	b(1,3)	a(4)	a(5)

La dipendenza DE->C ci permette di unificare b(1,3) con a(3) per l'attributo C nelle righe R3,R4,R5

	A	B	C	D	E
R1	a(1)	b(1,2)	b(1,3)	a(4)	b(1,5)
R2	a(1)	a(2)	b(1,3)	a(4)	b(2,5)
R3	b(3,1)	a(2)	a(3)	a(4)	a(5)
R4	b(4,1)	b(4,2)	a(3)	a(4)	a(5)
R5	a(1)	b(5,2)	a(3)	a(4)	a(5)

Determinazione di scomposizioni lossless joins

Esempio (2) (continua)

Notare che poiche' vale la dipendenza A->C e
si unifica anche il simbolo b(1,3) della colonna C nelle righe R1 e R2 con il simbolo a(3); quindi si ottiene:

	A	B	C	D	E
R1	a(1)	b(1,2)	a(3)	a(4)	b(1,5)
R2	a(1)	a(2)	a(3)	a(4)	b(2,5)
R3	b(3,1)	a(2)	a(3)	a(4)	a(5)
R4	b(4,1)	b(4,2)	a(3)	a(4)	a(5)
R5	a(1)	b(5,2)	a(3)	a(4)	a(5)

Infine la dipendenza CE->A ci permette di unificare i simboli b(3,1), b(4,1), a(1) per l'attributo A nelle righe R3, R4, R5. Il simbolo scelto come rappresentante e' a(1)

Determinazione di scomposizioni lossless joins

Esempio (2) (continua)

Otteniamo:

	A	B	C	D	E
R1	a(1)	b(1,2)	a(3)	a(4)	b(1,5)
R2	a(1)	a(2)	a(3)	a(4)	b(2,5)
R3	a(1)	a(2)	a(3)	a(4)	a(5)
R4	a(1)	b(4,2)	a(3)	a(4)	a(5)
R5	a(1)	b(5,2)	a(3)	a(4)	a(5)

Poiche' esiste una riga (R3) che ha tutti simboli a(i) la scomposizione ha la proprieta' di lossless join

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Determinazione di scomposizioni lossless joins

Esempio (3) :

- sia R=ABCD uno schema di relazione
- sia R1=AB R2=AD R3=BC
  una scomposizione di R
- date le seguenti dipendenze funzionali
  A->B A->D

determinare se la scomposizione ha la proprieta' di lossless join

Matrice iniziale:

	A	B	C	D
R1	a(1)	a(2)	b(1,3)	b(1,4)
R2	a(1)	b(2,2)	b(2,3)	a(4)
R3	b(3,1)	a(2)	a(3)	b(3,4)

Usando A->B otteniamo

	A	B	C	D
R1	a(1)	a(2)	b(1,3)	b(1,4)
R2	a(1)	a(2)	b(2,3)	a(4)
R3	b(3,1)	a(2)	a(3)	b(3,4)

Determinazione di scomposizioni lossless joins
Esempio (3) (continua):

Applicando la dipendenza A->D otteniamo di unificare b(1,4) e a(4) in a(4)

	A	B	C	D
R1	a(1)	a(2)	b(1,3)	a(4)
R2	a(1)	a(2)	b(2,3)	a(4)
R3	b(3,1)	a(2)	a(3)	b(3,4)

A questo punto non possiamo eseguire piu' modifiche pertanto la scomposizione non ha la proprieta' di essere lossless join

Invece la scomposizione in R1=ABC e R2=ACD
e' lossless join

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Determinazione di scomposizioni lossless joins

Teorema 4: L'algoritmo 2 determina correttamente se una scomposizione ha la proprieta' di lossless join.

L'algoritmo 2 puo' essere applicato a scomposizioni in un qualsiasi numero di schemi di relazioni.

Esiste un test piu' semplice che si applica al caso in cui la scomposizione sia in due schemi di relazioni:

Teorema 5: Data ρ = (R1,R2) scomposizione di uno schema R, e dato un insieme *F* di dipendenze funzionali, ρ ha la proprieta' di lossless join se e solo se vale una delle due condizioni:

(i) $(R1 \cap R2) \rightarrow (R1-R2)$

(ii) $(R1 \cap R2) \rightarrow (R2-R1)$.

Nota: non occorre che queste dipendenze siano in *F*; e' sufficiente che siano in *F*+

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Determinazione di scomposizioni lossless joins

Esempio di applicazione del teorema 5

R=ABC $\quad F = \{ A \text{->} B\}$

- La scomposizione di R in
  R1=AB e R2=AC ha la proprieta' di lossless join
  in quanto:
  $\{AB\} \cap \{AC\} = A$, $\{AB\} - \{AC\} = B$ e A->B vale

- La scomposizione di R in
  R1=AB e R2=BC non ha la proprieta' di lossless join
  in quanto:
  $\{AB\} \cap \{BC\} = B \qquad \{AB\} - \{BC\} = A \qquad \{BC\} - \{AB\} = C$
  ma B non determina funzionalmente ne' A ne' C

  come esempio consideriamo la relazione
  r={a1b1c1, a2b1c2} di schema R
  $\Pi_{AB}(r) = \{a1b1, a2b1\} \qquad \Pi_{BC}(r) = \{b1c1, b1c2\}$

  Il join naturale delle due proiezioni ci restituisce
  {a1b1c1, a1b1c2, a2b1c1, a2b1c2}

  (quindi non riotteniamo la relazione di partenza)

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Scomposizioni che preservano le dipendenze

- Abbiamo visto che e' importante per le scomposizioni avere la proprieta' di lossless join in quanto questo ci garantisce che qualsiasi relazione puo' essere riottenuta dalle sue proiezioni

- Un'altra importante proprieta' di una scomposizione di uno schema R in ρ=(R1,......, Rk) e' che l'insieme delle dipendenze *F* definito per R sia implicato dalle "proiezioni" di *F* sugli schemi R1,.....,Rk

- Dato *F* per un insieme di attributi D, e dato un insieme Z $\subseteq$ D, la proiezione di *F* su Z, denotata da $\Pi_Z(F)$, e' l'insieme di dipendenze X->Y in *F*+ tali che XY $\subseteq$ Z.
  (Notare che X->Y non deve necessariamente essere in *F* ; e' sufficiente che sia in *F*+)

- Una scomposizione ρ *preserva* un insieme di dipendenze *F* se l'unione di tutte le dipendenze $\Pi_{Ri}(F)$ (per i=1,....,k) implica logicamente tutte le dipendenze in *F*.

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Scomposizioni che preservano le dipendenze

- Il motivo per cui e' utile preservare le dipendenze e' che tali dipendenze sono dei vincoli di integrita'

- Consideriamo lo schema R=(CSZ)
  con dipendenze funzionali {CS->Z, Z->C}

  la scomposizione in R1=SZ e R2=CZ ha la proprieta' di lossless join; infatti

  $(\{SZ\} \cap \{CZ\}) \rightarrow (\{CZ\} - \{SZ\})$

  D'altra parte la proiezione di {CS->Z, Z->C} su SZ da solo le dipendenze banali che seguono dall'assioma di riflessivita' (cioe' S->S e Z->Z)

  mentre la proiezione su CZ da' le dipendenze banali piu' la dipendenza Z->C
  (quindi perdiamo la dipendenza CS->Z in quanto le dipendenze banali e la dipendenza Z->C non implicano la dipendenza CS->Z)

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Scomposizioni che preservano le dipendenze

Esempio di join che viola la dipendenza CS->Z

Consideriamo la seguente scomposizione:

R1=SZ e R2=CZ

R1

S	Z
s1	z1
s1	z2

R2

C	Z
c1	z1
c1	z2

Eseguiamo il join di R1 e R2

S	Z	C
s1	z1	c1
s1	z2	c1

Questo join viola la dipendenza CS->Z in quanto la relazione ottenuta come risultato ha due tuple che pur avendo lo stesso valore di C e S hanno valori diversi per Z

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Scomposizioni che preservano le dipendenze

- Un modo per controllare se una scomposizione ρ=(R1,......, Rk) preserva un insieme di dipendenze *F* e' il seguente (che deriva dalla definizione stessa di scomposizione che preserva le dipendenze):

  si calcola *F*+ e se ne esegue la proiezione su tutti gli Ri (i=1,...,k);
  se esegue l'unione delle dipendenze ottenute come proiezione e si controlla se questo insieme copre *F*

- Questo modo e' piuttosto costoso in quanto il numero di dipendenze contenute in *F*+ e' esponenziale rispetto alla dimensione di *F*

- Esiste un algoritmo che permette di controllare se una scomposizione preserva le dipendenze funzionali con un costo che e' polinomiale rispetto alla dimensione di *F*

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Scomposizioni che preservano le dipendenze

Algoritmo 3: Controllo della preservazione di un insieme di dipendenze funzionali

Input: Una scomposizione ρ=(R1,......, Rk) e un insieme di dipendenze funzionali *F*

Output: Decisione sulla preservazione delle dipendenze funzionali da parte di ρ

Metodo: Si definisce *G* come $\bigcup \Pi_{Ri}(F)$.

Notare che non calcoliamo *G*; semplicemente vogliamo determinare se copre *F*.

Per determinare se *G* copre *F*, dobbiamo considerare ogni X->Y in *F*, e determinare se X+, calcolato rispetto a *G*, contiene Y.

Per calcolare X+ senza avere a disposizione *G*, si considera ripetutamente quale e' l'effetto di eseguire la chiusura di X rispetto alle proiezioni di *F* sulle varie Ri.

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Scomposizioni che preservano le dipendenze

Metodo:(continua)

Dato uno schema Ri si definisce la *Ri-operazione* su un insieme di attributi Z rispetto ad un insieme di dipendenze *F* come la sostituzione di Z con

$Z \cup ((Z \cap Ri)^+ \cap Ri)$

dove la chiusura si esegue rispetto ad *F*

Questa operazione aggiunge a Z gli attributi A tali che $(Z \cap Ri) \rightarrow A$ e' in $\Pi_{Ri}(F)$

Si calcola X+ rispetto a *G* partendo con X, e ripetutamente eseguendo la *Ri-operazione* per ogni Ri

Se ad un qualche passo, nessuna *Ri-operazione* modifica l'insieme corrente di attributi, tale insieme e' X+

Formalmente l'algoritmo e':

Z = X
while changes to Z occur **do**
        for i=1 **to** k **do** $Z \cup ((Z \cap Ri)^+ \cap Ri)$

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Scomposizioni che preservano le dipendenze

Metodo:(continua)

Se Y e' un sottoinsieme dell'insieme Z ottenuto dai passi precedenti, allora X->Y e' in *G*

se ogni X->Y in *F* e' anche in *G*, allora la scomposizione preserva le dipendenze; altrimenti no.

Esempio:

R=(ABCD)

scomposizione R1=AB R2=BC R3=CD

F= { A->B, B->C, C->D, D->A}

(notare che in *F*+ ogni attributo determina funzionalmente ogni altro attributo)

Dobbiamo controllare che se facciamo la proiezione di *F* sulla scomposizione e poi eseguiamo l'unione delle proiezioni, tale unione copre *F*

$G = \Pi_{AB}(F) \cup \Pi_{BC}(F) \cup \Pi_{CD}(F)$

SCOMPOSIZIONE DI SCHEMI RELAZIONALI

Scomposizioni che preservano le dipendenze

Esempio (continua)

1) D->A

Z={D} dobbiamo applicare le *R-operazioni* relative a tutti gli schemi Ri

(i) applichiamo la *AB-operazione*

$\{D\} \cup ((\{D\} \cap \{A,B\})^+ \cap \{A,B\}) = \{D\}$

(quindi Z non e' modificato)

(ii) applichiamo la *BC-operazione*

$\{D\} \cup ((\{D\} \cap \{B,C\})^+ \cap \{B,C\}) = \{D\}$

(quindi Z non e' modificato)

(iii) applichiamo la *CD-operazione*

$$\begin{aligned} Z &= \{D\} \cup ((\{D\} \cap \{C,D\})^+ \cap \{C,D\}) \\ &= \{D\} \cup (\{D\}^+ \cap \{C,D\}) \\ &= \{D\} \cup (\{A, B, C, D\} \cap \{C,D\}) \\ &= \{C, D\} \end{aligned}$$

(iv) Si applica di nuovo la *AB-operazione* che non modifica Z

(v) Si applica la *BC-operazione*

$$\begin{aligned} Z &= \{C,D\} \cup ((\{C,D\} \cap \{B,C\})^+ \cap \{B,C\}) \\ &= \{C,D\} \cup (\{C\}^+ \cap \{B,C\}) \\ &= \{C,D\} \cup (\{A, B, C, D\} \cap \{B,C\}) \\ &= \{C,D\} \cup \{B,C\} = \{B, C, D\} \end{aligned}$$

Scomposizioni che preservano le dipendenze

Esempio (continua)

(vi) Si applica la *AB-operazione* ottenendo

Z = {A, B, C, D}

Dopo di che non non e' piu'possibile modificare Z

Pertanto dovendo verificare D->A, abbiamo che rispetto a *G*, {D}+={A, B, C, D}
quindi poiche' {D}+ include A abbiamo che *G* |= D->A
(cioe' abbiamo che la dipendenza D->A e' implicata logicamente da *G*)

E' facile vedere che anche tutte le altre dipendenze sono preservate. Quindi questa scomposizione preserva le dipendenze

Teorema 6: L'algoritmo 3 determina correttamente se X->Y e' in *G*+.

FORME NORMALI

Forma normale di Boyce-Codd

- Uno schema di relazione R con dipendenze *F* e' in *forma normale di Boyce-Codd* se:
  per ogni X->A che vale per R, e A non e' in X, allora X e' una superchiave di R (cioe' X e' la chiave o contiene la chiave)

- In altre parole: tutte le dipendenze non banali sono quelle in cui una chiave determina funzionalmente uno o piu' degli altri attributi

Esempio Lo schema CSZ con dipendenze CS->Z e Z->C non e' in forma normale di Boyce-Codd.
Infatti, vale la dipendenza Z->C, ma Z non e' una chiave di CSZ, ne' tantomeno contiene la chiave

(Nota: per determinare se Z e' chiave rispetto alla dipendenze funzionali date, occorre calcolare Z+ rispetto alle dipendenze date e poi vedere se Z+ contiene tutti gli altri attributi;
Z+ = {Z,C} e quindi non contenendo S, Z non e' chiave)

FORME NORMALI

Forma normale di Boyce-Codd

Esempio: gli schemi
Members (Name, Address, Balance)
Orders (Order_no, Name, Item, Quantity)

con le dipendenze funzionali

Name -> Address Balance
Order_no -> Name Item Quantity

sono in forma normale di Boyce-Codd, poiche' le uniche dipendenze funzionali sono quelle che hanno come lato sinistro la chiave

FORME NORMALI

Terza forma normale

- In alcuni casi la forma normale di Boyce-Codd e' troppo forte, in quanto puo' non preservare le dipendenze funzionali

- La terza forma normale ha quasi gli stessi benefici della forma normale di Boyce-Codd, ma preserva le dipendenze funzionali

Definizione: Un attributo A di uno schema di relazione e' un attributo *primo* se e' elemento di una qualche chiave di R (nota: R puo' avere piu' chiavi). Se A non e' elemento di una qualche chiave A e' *nonprimo*.

Esempio: CSZ con *F* = {CS->Z, Z->C}
tutti gli attributi sono primi in quanto sia CS che SZ sono chiavi; pertanto ogni attributo e' contenuto in almeno una chiave

Esempio: ABCD con *F* = {AB->C, B->D, BC->A}
le sole chiavi sono AB e BC, quindi
A,B,C sono primi, mentre D e' nonprimo

FORME NORMALI

Terza forma normale

Uno schema di relazione R con dipendenze *F* e' in *terza forma normale* (3NF) se:

per ogni X->A che vale per R, tale che A non e' in X, allora:

(i) X e' una superchiave di R (cioe' X e' la chiave o contiene la chiave)

oppure

(ii) A e' primo

- La differenza fondamentale e' che nella terza forma normale si ammettono dipendenze funzionali che non abbiano una chiave sul lato sinistro, purche' sul lato destro si abbia un attributo primo

Esempio: CSZ con *F* = {CS->Z, Z->C}

e' in terza forma normale in quanto tutti gli attributi sono primi

Terza forma normale

Esempio: SAIP con F ={ SI->P, S->A}

L'unica chiave per questa relazione e' SI

Gli attributi primi sono {S,I}

La dipendenza SI->P non viola la 3NF in quanto verifica la condizione (i)

La dipendenza S->A viola la 3NF in quanto S non e' superchiave e A e' nonprimo

Esempio: Schema R = (S, I, D, M)

S = Store I=Item D=Department M=Manager Dipendenze funzionali:

SI->D ogni prodotto e' venduto da al piu' un

dipartimento per negozio

SD->M ogni dipartimento ha un solo manager

La chiave e' SI: si determina calcolando (SI)+

La dipendenza SD->M viola la 3NF in quanto SD non e' una superchiave per R (in quanto non include la chiave); inoltre M e' nonprimo.

FORME NORMALI

Terza forma normale

Se una dipendenza X->A non verifica la 3NF vuol dire che o:

(i) X e' un sottoinsieme proprio della chiave
oppure
(ii) X non e' sottoinsieme proprio di alcuna chiave

Nel caso (i) si dice che X->A e' una *dipendenza parziale* :

cioe' A dipende non da tutta la chiave ma da una sottoparte della chiave

Nel caso (ii) si dice che X->A e' una *dipendenza transitiva:*

- questo vuol dire che se Y e' la chiave della relazione, esiste una catena di dipendenze non banali Y->X->A
- la catena e' non banale in quanto
  (a) X non e' un sottoinsieme di Y
  (b) A non e' in X per definizione di 3NF
  (c) A non e' in Y in quanto e' nonprimo

FORME NORMALI

Terza forma normale

Esempio: Schema R = (S, I, D, M)

Dipendenze funzionali:
SI->D SD->M
La relazione R non e' in 3NF
Non ha dipendenze parziali in quanto non c'e' alcun sottoinsieme proprio che determina funzionalmente M e D

Ha una dipendenza transitiva:
SI->D quindi SI->SD quindi SD->M
SI->SD->M
La dipendenza SD->M viola la 3NF in quanto SD non e' una superchiave per R (in quanto non include la chiave); inoltre M e' nonprimo.

Uno schema di relazione R e' in *seconda forma normale* (2NF) se non ha dipendenze parziali (uno schema di relazione in 2NF puo' tuttavia avere dipendenze transitive)

FORME NORMALI

Terza forma normale - motivazioni

- Se esiste una dipendenza parziale Y->A, dove X e' la chiave e Y e' un sottoinsieme proprio di X, allora in ogni tupla che ha un valore di X, e' necessario che la stessa associazione tra Y e A appaia

- Esempio: supplier=(sname, saddress,item, price)
  chiave (sname, saddress)
  esiste la dipendenza parziale
  sname->saddress

  Questa dipendenza ci obbliga a ripetere lo stesso valore di saddress per ogni item fornito dallo stesso fornitore
  (provocando le anomalie viste)

FORME NORMALI

Terza forma normale - motivazioni

- Se abbiamo una dipendenza transitiva X->Y->A non possiamo associare un valore di Y ad uno di X, a meno che ci sia un valore assegnato ad A

- Questo causa anomalie di inserzione e cancellazione in quanto non possiamo inserire una associazione X-a-Y, senza inserirne una da Y-a-A

- Esempio: Schema R = (S, I, D, M)

  Dipendenze funzionali:
  SI->D SD->M

  Non possiamo inserire il fatto che il dipartimento 10 vende cappelli a Bloomingdales se il dipartimento non ha un manager

FORME NORMALI

BC-NF e 3NF

- La BC-NF previene alcune anomalie che la 3NF non previene
  Per esempio: nello schema R=(CSZ) con dipendenze
  CS->Z e Z->C
  non e' possibile inserire un valore per Z se non si conosce il corrispondente valore per C

- La BC-NF ha il problema che puo' generare delle scomposizioni che non preservano le dipendenze (questo non succede con la 3NF)

- Pertanto se la scomposizione BC-NF non preserva le dipendenze, e' necessario usare la 3NF

- Notare che se si general lo schema concettuale in ER e quindi si applicano le regole di trasformazione, si ottengono quasi sempre degli schemi di relazione in 3NF (a meno che non ci siano delle dipendenze tra gli attributi di una stessa entita')
  Entita' Employee con attributi
  (ssn, name,....., office, office-phone,..........)
  se ad esempio office->office-phone esiste una dipendenza transitiva

FORME NORMALI

Scomposizione lossless join in BC-NF

- Qualsiasi schema di relazione ha una scomposizione BC-NF che ha la proprieta' di lossless join

- Qualsiasi schema di relazione ha un scomposizione 3NF che ha la proprieta' di lossless join e preserva le dipendenze

- D'altra parte e' possibile che non esista, per un dato schema di relazione, una scomposizione in BC-NF che preserva le dipendenze
  Esempio: R=(CSZ) *F* = {CS->Z, Z->C}
  la scomposizione in R1=SZ e R2=CZ perde la dipendenza funzionale CS->Z

FORME NORMALI

Scomposizione lossless join in BC-NF

Lemma 6:

a) Sia R uno schema di relazione con un insieme di dipendenze F . Sia ρ =(R1,....,Rk) una scomposizione di R con la proprieta' di lossless join rispetto ad F .
Per un certo i, sia $Fi = \Pi_{Ri}(F)$, e sia
σ=(S1,......,Sm) una scomposzione di Ri con proprieta' di lossless join rispetto ad Fi , allora la scomposizione di R in
(R1,.......,Ri-1,S1,..........,Sm, Ri+1,.........,Rk)
la proprieta' di lossless join rispetto ad F .

b) Siano R, F, ρ come in (a), e sia
τ = (R1,......,Rk, Rk+1,........,Rn) una scomposizione di R in un insieme di schemi di relazioni che include gli schemi di ρ.
Allora anche τ la proprieta' di lossless join rispetto ad F .

FORME NORMALI

Scomposizione lossless join in BC-NF

Algoritmo 4: Determina scomposizioni BC-NF con proprieta' di lossless join

Input: Uno schema di relazione R e un insieme di dipendenze funzionali *F*

Output: Una scomposizione di R con proprieta' di lossless join, tale che ogni schema ottenuto dalla scomposizione e' in BC-NF rispetto alle proiezioni di *F* su tali schemi

Metodo: Si costruisce iterativamente una scomposizione ρ per R

- Ad ogni passo ρ ha la proprieta' di lossless join rispetto ad *F*
- Inizialmente, ρ contiene solo R
- (*) Se S e' uno schema in ρ e S non e' in BC-NF,
  sia X->A una dipendenza che vale per S, tale che
  X non e' una superchiave di S e A non e' in X
  si sostituisce S in ρ con due schemi S1 e S2 tali che
  S1 consiste degli attributi di X e di A
  S2 consiste degli attributi di S meno A
  (si calcolano le proiezioni di *F*+ su S1 e su S2)

- Si ripete (*) fino a che ogni schema in ρ e' in BC-NF

Scomposizione lossless join in BC-NF

- L'algoritmo 4 e' ad ogni passo scompone uno schema S in due schemi S1 e S2 che hanno la proprieta' di lossless join

  Questo consegue dal teorema 5 (condizione semplificata per il lossless join)
  $S1 \cap S2 = X \quad S1\text{-}S2 = A$
  e quindi $(S1 \cap S2) \rightarrow (S1\text{-}S2)$
  Per il Lemma 6(a), se ρ e' lossless join, sostituendo in ρ S con S1 e S2, ρ rimane lossless join

  (notare che inizialmente ρ e' lossless join in quanto contiene un solo schema relazionale, che e' quello di partenza)

FORME NORMALI

Scomposizione lossless join in BC-NF

Esempio:

R=CTHRSG

dove C=course, T=teacher , H=hour, S = student
G=grade

Valgono le seguenti dipendenze funzionali:

C->T ogni corso ha un solo insegnante

HR->C un solo corso puo' essere tenuto in una data aula ad una data ora

HT->R un docente non puo' essere sono in due aule contemporaneamente

CS->G ogni studente ha un solo voto per ogni esame

HS->R uno studente non puo' essere contemporaneamente in due aule

Questo schema relazionale ha una sola chiave HS

Per scomporre R in BC-NF consideriamo la dipendenza CS->G, che non verifica la condizione di essere una superchiave della relazione:

applicando l'algoritmo scomponiamo R in due schemi S1=(CSG) e S2=(CTHRS)

(nota: data una dipendenza X->A, S1=X U A S2=R-A)

Si calcolano le proiezioni di *F*+ su CSG e CTHRS

$\Pi_{CSG}(F)$ = {CS->G} chiave CS

$\Pi_{CTHRS}(F)$ = {C->T, HR->C, TH->R, HS->R} chiave HS

FORME NORMALI

Scomposizione lossless join in BC-NF

Esempio: (continua)

- CSG e' in BC-NF
- CTHRS non lo e'; usiamo la dipendenza C->T per scomporre in CT e CHRS

  $\Pi_{CT}(F) = \{C\text{->}T\}$ chiave C

  $\Pi_{CHRS}(F) = \{CH\text{->}R, HS\text{->}R, HR\text{->}C\}$ chiave HS

  (notare che CH->R e' necessaria nella proiezione su CHRS ma non in CTHRS in quanto in quest'ultima e' implicata da C->T e TH->R)

- CT e' in BC-NF

- una ulteriore scomposizione di CHRS usando CH->R, genera

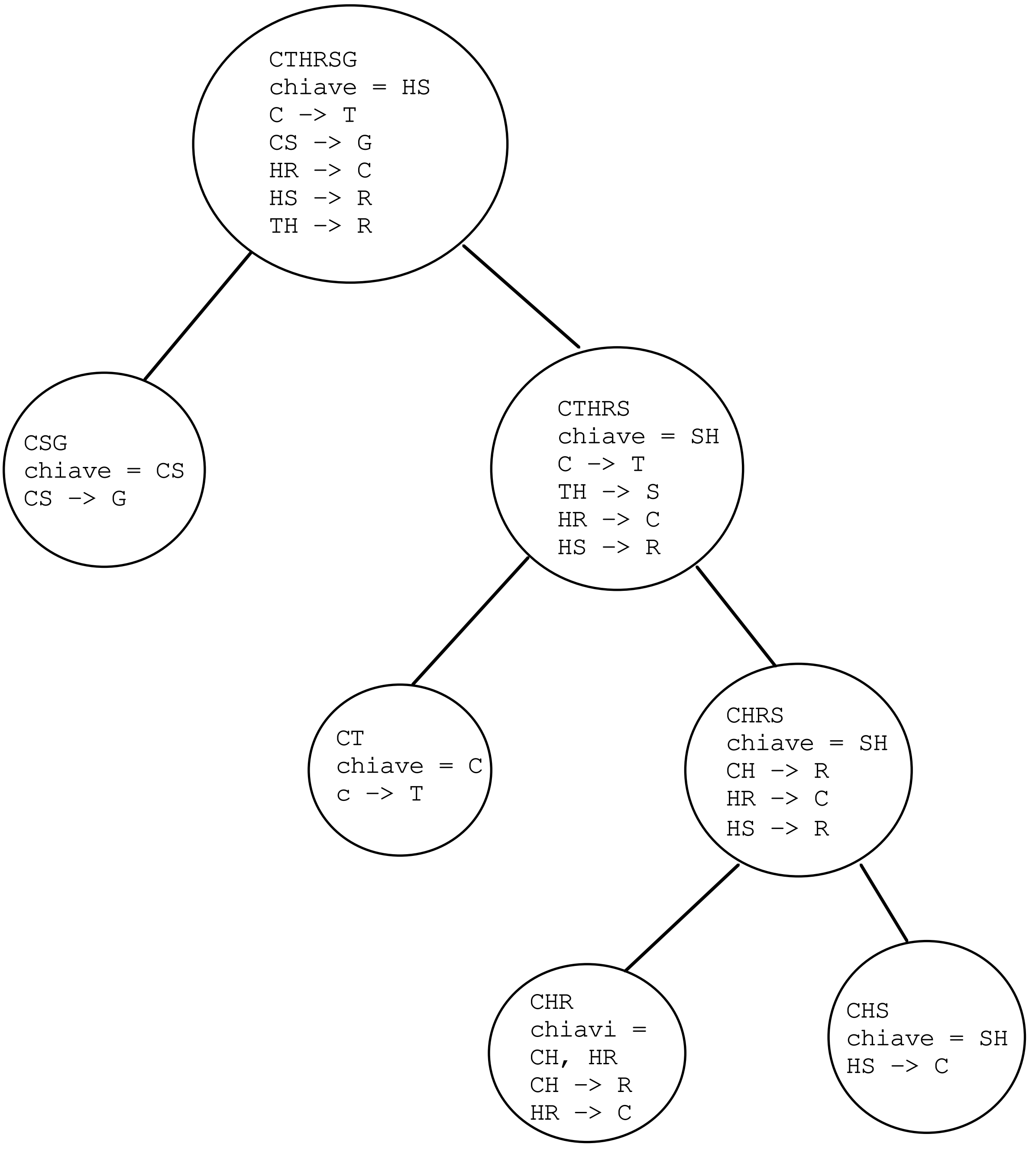
  CHR con {CH->R, HR->C}, con chiavi {CH, HR}

  CHS con {HS->C}, con chiave SH

- La scomposizione ottenuta e' in BC-NF

FORME NORMALI

Scomposizione lossless join in BC-NF
Albero di scomposizione



FORME NORMALI

Scomposizione lossless join in BC-NF

La scomposizione finale e':

1. CSG: contiene il voto per studente per corso
2. CT: contiene il docente per ogni corso
3. CHR: contiene per ogni corso l'orario e l'aula
4. CHS: contiene per ogni studente i corsi e l'orario

Notare che si possono ottenere scomposizioni diverse a seconda delle dipendenze che si usano e in che ordine si esaminano:

> per esempio consideriamo l'ultimo passo
> CHRS e' stata scomposta in
> CHR e CHS essendo stata usata la dipendenza CH->R
>
> usando invece HR->C si otterebbe
> CHR e HRS

FORME NORMALI

Scomposizione lossless join in BC-NF

- Un altro problema e' che la dipendenza TH->R non e' preservata

- La proiezione di *F* su CSG, CT, CHR, e CHS che e' rappresentata dal seguente insieme minimo di copertura

  CS->G HR->C C->T
  CH->R HS->C

  non implica la dipendenza TH->R

  Per esempio la relazione CTHRSG

C	T	H	R	S	G
c1	t	h	r1	s1	g1
c2	t	h	r2	s2	g2

non soddisfa TH->R, nonostante le sue proiezioni su CSG, CT, CHR, e CHS verifichino tutte le dipendenze ottenute dalla proiezione

FORME NORMALI

Scomposizione in 3NF

- Non e' sempre possibile ottenere una scomposizione in BC-NF che preservi le dipendenze

- E' invece sempre possibile determinare una scomposizione in 3NF che preserva le dipendenze

Algoritmo 5 : Scomposizione in 3NF con preservazione delle dipendenze funzionali

Input: Uno schema di relazione R e un insieme *F* (che sia un cover minimo) di dipendenze funzionali

Output: Una scomposizione loss-less join di R tale che ogni schema e' in 3NF rispetto ad *F* e ogni dipendenza e' preservata

FORME NORMALI

Scomposizione in 3NF

Metodo: i=0; S=$\varnothing$

for each dipendenza funzionale X->A in F **do**

begin i:=i+1; Ri := XA; S := S $\cup$ $\{$Ri$\}$; **end**

Optionally: le relazioni di schema

(XA1), (XA2), ..., (XAn)

ottenute da dipendenze rispettivamente

della forma X->A1, X->A2, ..., X->An

possono essere sostituite dalla relazione

di schema (X A1 A2 .... An)

if nessuno schema Rj ($1 \leq j \leq i$) contiene una chiave

candidata **then begin**

i:=i+1;

Ri:= una qualsiasi chiave candidata di R

S := S $\cup$ $\{$Ri$\}$; **end**

for each Ri, Ri $\in$ S, **if exists** Rj $\in$ S, i $\neq$ j,

such that schema(Rj) $\supset$ schema(Ri), **then** S:=S-{Ri};

return (S)

FORME NORMALI

Scomposizione in 3NF

Esempio:
R=CTHRSG
F = {C->T, HR->C, HT->R,CS->G, HS->R}

chiave HS

Prima parte algoritmo:

R1=CT R2=CHR R3=HRT

R4=CGS R5=HRS

Seconda parte: si controlla se esiste almeno uno schema che contiene la chiave candidata

R5 contiene la chiave candidata, quindi non occorre creare una relazione aggiuntiva

FORME NORMALI

Scomposizione in BC-NF e 3NF

Esempio 1:

S = (J,K,L) con *F* = {JK->L, L->K}

Scomposizione in BC-NF
Applicando l'algoritmo 4, abbiamo che S non e' in BC-NF (a causa della dipendenza L->K);
applicando la dipendenza L->K
otteniamo: R1=(LK), R2=(JL)
questo schema non preserva la dipendenza JK->L

Scomposizione in 3NF
Lo schema S e' gia' in 3NF in quanto esiste una dipendenza non banale che ha come lato sinistro un attributo che non e' superchiave (i.e. L->K);
d'altra parte essendo K un attributo primo verifica le condizioni di 3NF

Esempio 2:

R = (branch-name, assets, branch-city, loan-number,
customer-name, amount)

branch-name -> assets
branch-name -> city
loan-number -> amount
loan-number -> branch-name
la chiave e' (loan-number, customer name)

Lo schema R non e' in BC-NF

1) Consideriamo una dipendenza che viola le BC-NF

branch-name -> assets

R e' sostituita da

R1 = (branch-name, assets)
R2 = (branch-name, branch-city, loan-number,
customer-name, amount)

- R1 e ' in BC-NF con chiave branch-name
- R2 non lo e'

Esempio 2: (continua)

2) Consideriamo la dipendenza

branch-name-> branch-city

R2 e' sostituita da

R3 = (branch-name, branch-city)

R4 = (branch-name, loan-number, customer-name, amount)

- R3 e' in BC-NF con chiave branch-name
- R4 non lo e'

3) Consideriamo la dipendenza

loan-number -> amount

R4 e' sostuita da

R5 = (loan-number, amount)

R6 = (branch-name, loan-number, customer-name)

R5 e' in BC-NF con chiave loan-number

R6 non lo e'

4) Consideriamo la dipendenza

loan-number -> branch-name

R6 e' sostituita da

R7 = (loan-number, branch-name)

R8 = (loan-number, customer-name)

Schema finale: R1, R3, R5, R7, R8

FORME NORMALI

Scomposizione in BC-NF e 3NF

Esempio 3:

R = (branch-name, loan-number, amount,
account-number, balance, customer-name)

loan-number -> amount

loan-number -> branch-name

account-number -> balance

account-number -> branch-name

la chiave e' (loan-number, account-number, customer-name)

Applicando l'algoritmo si ottiene il seguente schema:

R1 = (loan-number, amount)

R2 = (loan-number, branch-name)

R3 = (account-number, balance)

R4 = (loan-number, account-number, customer-name)

Per determinare se preserva le dipendenze si applica l'algoritmo 3

FORME NORMALI

Scomposizione in BC-NF e 3NF

Esempio 3: (continua)

1) loan-number -> amount (LN->A)

Z = {L-N}

(i) applichiamo la R1-operazione

{L-N} U (({L-N} ∩ {L-N, A})+ ∩ {L-N, A}) =

{L-N, A}

(ii) applichiamo la R2-operazione

{L-N, A} U (({L-N, A} ∩ {L-N, B-N})+

∩ {L-N, B-N}) = {L-N, A, B-N}

La dipendenza LN->A e' preservata

2) loan-number -> branch-name e' anche preservata (passo (ii) precedente)

3) account-number -> balance (A-N->B)

Z = {A-N}

(i) applichiamo la R1-operazione

{A-N} U (({L-N} ∩ {L-N, A})+ ∩ {L-N, A}) =

{A-N}

(ii) applichiamo la R2-operazione Z={A-N}

(iii) applichiamo la R3-operazione

Z={A-N, B}

Esempio 3: (continua)

4) account-number -> branch-name (A-N->B-N)

Z = {A-N}

(i) applichiamo la R1-operazione

{A-N} U (({L-N} ∩ {L-N, A})+ ∩ {L-N, A}) =

{A-N}

(ii) applichiamo la R2-operazione

{A-N} U (({A-N} ∩ {L-N, B-N})+

∩ {L-N, B-N}) = {A-N}

(iii) applichiamo la R3-operazione

{A-N} U (({A-N} ∩ {A-N, B})+ ∩ {A-N, B}) =

{A-N} U ((A-N)+ ∩ {A-N, B}) =

{A-N} U ((A-N, B, B-N) ∩ {A-N, B}) =

{A-N, B}

(iv) applichiamo la R4-operazione

{A-N,B} U (({A-N,B} ∩ {L-N, A-N,C-N})+

∩ {L-N, A-N,C-N}) =

{A-N,B} U ({A-N} + ∩ {L-N, A-N,C-N}) =

{A-N,B} U ({A-N, B, B-N})∩ {L-N, A-N,C-N})=

{A-N,B} U (A-N) ={A-N, B}

Si determina che la dipendenza A-N->B-N non e' preservata

FORME NORMALI

Scomposizione in BC-NF e 3NF

Esempio 3: (continua)

- Pertanto la scomposizione ottenuta non preserva le dipendenze
- Determiniamo una scomposizione in 3NF

  Parte 1 algoritmo

  R1 = (loan-number, amount)

  R2 = (loan-number, branch-name)

  R3 = (account-number, balance)

  R4 = (account-number, branch-name)

  Parte 2 algoritmo

  Determiniamo se la chiave e' in uno degli schemi R1, R2, R3, R4

  poiche' non lo e', aggiungiamo uno schema che la contenga

R5 = (loan-number, account-number, customer-name)