Normalizzazione

1

Normalizzazione

- Lo scopo della normalizzazione è di progettare dei "buoni" schemi di basi di dati; gli schemi normalizzati evitano specifiche *anomalie*
- Se si usano metodologie basate su una definizione in termini del modello ER dello schema e successiva traduzione di questo in termini del modello relazionale gran parte delle anomalie non si verificano
- Molto spesso la normalizzazione conduce a definire più relazioni con meno attributi, si noti come questo possa portare a delle inefficienze nelle prestazioni (join)

2

Ridondanze e anomalie

impiegato	stipendio	progetto	budget	funzione
Rossi	2	biella	300	tecnico
Verdi	3	valvola	500	progettista
Verdi	3	albero	1500	progettista
Neri	7	albero	1500	direttore
Neri	7	valvola	500	consulente
Neri	7	biella	300	consulente
Mori	6	biella	300	direttore
Mori	6	albero	1500	progettista
Bianchi	6	albero	1500	progettista
Bianchi	6	biella	300	progettista

3

Ridondanze e anomalie

1) ridondanza:
- si ripete più volte la notizia che un impiegato percepisce un certo stipendio
- si ripete più volte che un progetto ha un certo budget
- i valori di progetto e di impiegato si ripetono e quindi non possono singolarmente essere presi come chiave
- la chiave è (progetto, impiegato): non si hanno ripetizioni di informazioni associate al legame progetto-impiegato (es. funzione)

4

Ridondanze e anomalie

2) aggiornamento:
- poiché si ripete più volte la notizia che un impiegato percepisce un certo stipendio, se lo stipendio viene aggiornato questo deve essere fatto su tutte le tuple che riguardano un certo impiegato
- poiché si ripete più volte che un progetto ha un certo budget, se il budget viene aggiornato lo si deve fare su tutte le tuple che riguardano un certo progetto

5

Ridondanze e anomalie

3) cancellazione:
- supponendo che un impiegato lasci l'azienda o non partecipi a progetti rischiamo di perdere i dati sui progetti se era l'ultimo impiegato del progetto
- analogamente per i dati degli impiegati se un progetto viene eliminato
- se la chiave è (progetto, impiegato) in entrambi i casi di eliminazione si potrebbero avere valori nulli nella chiave

6

Ridondanze e anomalie

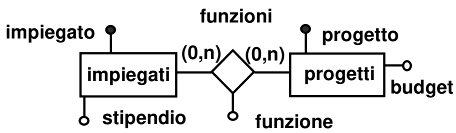
4) inserimento:
- poichè la chiave è (progetto, impiegato) non è possibile inserire i dati di un impiegato se non è stato assegnato ad almeno un progetto, analogamente per un nuovo progetto a cui non è stato ancora assegnato un impiegato
- accettare un inserimento di (progetto) o (impiegato) vuol dire inserire valori nulli (incompatibili con la chiave)

Ridondanze e anomalie

- Le anomalie viste non si presentano in genere se si è seguita la prassi corretta di progettazione: prima lo schema ER e poi la traduzione in schema relazionale
- può però succedere che carenze di specifiche o errori di schematizzazione possano portare a relazioni con anomalie
- i casi sono invece più frequenti quando si esaminano vecchi DB scarsamente documentati o si cerca di intuire la natura dei dati da documenti che sintetizzano le informazioni su moduli cartacei

Ridondanze e anomalie

- ER corretto



- traduzione in relazionale
  impiegati (<u>impiegato</u>,stipendio)
  progetti (<u>progetto</u>,budget)
  lavora (<u>impiegato</u>,<u>progetto</u>,funzione)

Dipendenze funzionali

- La dipendenza funzionale è un vincolo di integrità per il modello relazionale
- nel dominio applicativo dell'esempio un impiegato ha un unico stipendio
  - nella relazione, ogni volta che in una tupla compare un certo impiegato lo stipendio è sempre lo stesso
- possiamo dire che il valore dell'impiegato determina il valore dello stipendio, cioè:
  - esiste una funzione che associa ad ogni valore nel dominio impiegato uno ed un solo valore nel dominio stipendio

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 $X \rightarrow Y$ se per qualunque *r* stato (o estensione) di R non è possibile che due tuple in *r* che hanno valori uguali per gli attributi in X, abbiano valori diversi per gli attributi in Y
  (cioè se $t1, t2 \in r$ e t1.X=t2.X, allora t1.Y=t2.Y)
- nel nostro esempio:
  impiegato → stipendio   progetto → budget
  impiegato progetto → funzione

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 saldo e che ogni cliente abbia nome diverso. Queste assunzioni sono formalizzate dalla seguente dipendenza funzionale

Name → Address Balance

Dipendenze funzionali - Esempio

- Nella relazione Orders possiamo assumere che il numero di ordine determini tutto, poiché 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

13

Dipendenze funzionali - Esempio

- Una domanda è 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 è valida

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

14

Dipendenze funzionali - Esempio

- 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

15

Dipendenze funzionali - Esempio

- Altre dipendenze non banali sono derivate usando le dipendenze che abbiamo già 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

16

Dipendenze funzionali - Chiusura

- 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 e si indica con

  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, indicata come F+, è definita come l'insieme delle dipendenze funzionali implicate da *F*

  F+ = {X → Y | *F* |= X → Y}

17

Dipendenze funzionali - Chiusura

Esempio

- Nota: nel seguito si usa la notazione compatta A → B C per indicare A →B e A → C

- 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
  - X contiene A,
  - X contiene B ma non A e Y non contiene A,
  - X → Y è la dipendenza C → C

- F+ contiene ad esempio ABC → BC, AB → BC, A → C, BC → B

18

Dipendenze funzionali - Chiusura

- è stato definito un insieme di regole di inferenza che permettono di calcolare tutte le dipendenze in *F+*
- tali regole sono *sound* e *complete*, cioè permettono di ottenere tutte e sole le dipendenze in *F+*
- questo insieme di regole è chiamato *assiomi di Armstrong* (dal nome della persona che le ha definite nel 1974)

19

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: (*riflessività*) Se $Y \subseteq X \subseteq D$, allora $X \rightarrow Y$ è logicamente implicata da *F*
    - questa regola genera *dipendenze banali*, cioè dipendenze in cui la parte destra è contenuta nella parte sinistra; l'uso di questa regola non dipende da *F*
  - A2: (*additività*) Se $X \rightarrow Y$, e $Z \subseteq D$, allora $XZ \rightarrow YZ$ è implicata da *F*
  - A3: (*transitività*) Se $X \rightarrow Y$ e $Y \rightarrow Z$, allora $X \rightarrow Z$ è implicata da *F*

20

Regole derivate da Armstrong

- è possibile derivare altre regole dagli assiomi di Armstrong

a) Regola di unione

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

b) Regola di pseudotransitività

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

c) Regola di decomposizione

Se $Z \subseteq Y$, $X \rightarrow Y \models X \rightarrow Z$

21

Chiavi

- La nozione vista di chiave può essere formalizzata utilizzando le dipendenze funzionali
- Dato uno schema R con attributi A1 A2.....An e un insieme di dipendenze funzionali *F* , un X sottoinsieme di R si dice *chiave* di R se:
  - X → A1 A2.....An è in *F+*
    cioè la dipendenza di tutti gli attributi da X o è data o segue logicamente dalle dipendenze funzionali date
  - non esiste Y sottoinsieme proprio di X tale che
    Y → A1 A2.....An è in *F+*
    (condizione di minimalità)
- Si usa il termine *superchiave* per indicare un soprainsieme della chiave

22

Chiavi - Esempio

- Persona(CodiceFiscale, Nome, Cognome, DataNascita)
- F={CodiceFiscale → Nome Cognome DataNascita}

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

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

23

Chiavi - Esempio

- R(A,B,C) e *F* = {A→B, B→C}
- La chiave è A

  infatti A→A, A→B, A→C (per la proprietà transitiva) o equivalentemente A→ABC
- inoltre non esiste un sottoinsieme degli attributi di R non contenente A tale che

  X→ABC

24

Chiusura di un insieme di attributi

- Dato *F* insieme di dipendenze funzionali su un insieme di attributi D e X sottoinsieme di D la chiusura di X rispetto a *F* è denotata da X+ ed è definita come l'insieme di attributi {A / F |= X→A} = {A / X→A segue da F in base agli assiomi di Armstrong}
- X→Y ∈ *F*+ sse Y ∈ X+ rispetto ad F

Chiusura di un insieme di attributi

- il calcolo di *F+* è molto costoso (esponenziale nel numero di attributi dello schema nel caso peggiore)
- calcolare la chiusura transitiva rispetto ad un insieme di attributi X è meno costoso
- spesso quello che ci interessa non è calcolare tutto F+ ma solo verificare se contiene una certa dipendenza
- Algoritmo per il 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 ⊆ D
- *Output:* X+, chiusura di X rispetto ad *F*

Chiusura di un insieme di attributi

- *Approccio* Si calcola una sequenza di insiemi di attributi X(0), X(1),....... usando le seguenti regole
  - X(0) è X
  - X(i+1) è X(i) più un insieme di attributi A tali che:
    - c'è una dipendenza Y→Z in *F*
    - A è in Z
    - Y ⊆ X(i)
- poiché X=X(0) ⊆ ... ⊆ X(i) ⊆ … ⊆ D e D è 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)
- X+ è X(i) per un i tale che X(i)=X(i+1)

Chiusura di un insieme di attributi - Esempio

- Sia *F* il seguente insieme

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

- X(0)=BD
- X(1): uso dipendenza D→EG
  X(1)=BDEG
- X(2): uso dipendenza BE→C
  X(2)=BCDEG
- X(3): considero C→A, BC→D, CG→BD e CE→AG
  X(3)=ABCDEG
- (BD)+=X(3)

Equivalenza di insiemi di dipendenze

- Siano *F* e *G* insiemi di dipendenze funzionali, si dice che *F* e *G* sono equivalenti se *F+=G+*
- Per determinare l'equivalenza si procede come segue:
  - per ogni dipendenza Y→Z in *F*, si testa se Y→Z è in *G+* usando l'algoritmo visto per calcolare Y+ e quindi verificando se Z ⊆ Y+
  - se una qualche dipendenza Y→Z in *F* non è in *G+*, sicuramente *F+≠G+*
  - si verifica il viceversa determinando se ogni dipendenza funzionale in *G* è in *F+*

Insiemi di dipendenze minimali

- Un insieme di dipendenze *F* è minimale se:
  - il lato destro di ogni dipendenza in *F* è un singolo attributo
  - per nessuna X→A in *F* l'insieme *F* \ {X→A} è equivalente a *F*
  - per nessuna X→A in *F* e nessun sottoinsieme Z di X l'insieme *F* \ {X→A} U {Z→A} è 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
  - poiché sul lato destro c'è un solo attributo in base alla condizione (1), sicuramente non c'è nessun lato destro ridondante
- Ogni insieme di dipendenze *F* è equivalente ad un insieme minimale *F'*

Insiemi di dipendenze minimali

- Per determinare l'insieme minimale si procede come segue:
  - 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
  - si considera ogni dipendenza in *F'* che abbia più di un attributo sul lato sinistro, se è possibile eliminare un attributo dal lato sinistro e avere ancora un insieme di dipendenze equivalente, si elimina tale attributo (approccio: B in X è ridondante rispetto alla dipendenza X→A se (X\{B})+ calcolata rispetto a *F'* contiene A)
  - sia *F"* l'insieme generato, si considera ogni dipendenza X→A in *F"* in un qualche ordine e se *F"*\ {X→A} è equivalente a *F"*, si cancella X→A da *F"* (approccio: si calcola X+ rispetto a *F"*\{X→A} e se include A, allora X→A è ridondante)

Insiemi di dipendenze minimali

- Notare che al passo (2) l'ordine in base a cui le dipendenze funzionali sono esaminate può influire sugli insiemi minimali generati:

  A→B A→C B→A C→A B→C

  è 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 di dipendenze minimali

- Sia *F* il seguente insieme

  AB→C C→A BC→D ACD→B

  D→EG BE→C CG→BD CE→AG

- semplificando i lati destri otteniamo

  AB→C C→A BC→D ACD→B D→E D→G

  BE→C CG→B CG→D CE→AC E→G

- notiamo che:
  - CE→A è ridondante in quanto è implicato da C→A
  - CG→B è ridondante in quanto CG→D, C→A, e ACD→B implicano CG→B come può essere controllato calcolando (CG)+
  - inoltre ACD→B può essere sostituito da CD→B poiché C→A
- risultato:

  AB→C C→A BC→D CD→B D→E D→G

  BE→C CG→D CE→G

Insiemi di dipendenze minimali

- è possibile ottenere un altro insieme minimale eliminando:

  CE→A, CG→D e ACD→B

- l'insieme ottenuto è:

  AB→C C→A BC→D D→E

  D→G BE→C CG→B CE→G

Forma normale di Boyce-Codd

- Uno schema di relazione R con dipendenze F è in forma normale di Boyce-Codd (BCNF) se:
  - per ogni X→ A in F, tale che A non è in X, allora X è una superchiave di R (cioè X è la chiave o contiene la chiave)
- In altre parole: tutte le dipendenze non banali sono quelle in cui una chiave determina funzionalmente uno o più degli altri attributi

Forma normale di Boyce-Codd

Nel nostro esempio, la chiave è impiegato, progetto

- impiegato, progetto → funzione

  rispetta la BCNF,
- mentre

  impiegato → stipendio e

  progetto → budget

  sono dipendenze parziali che causano anomalie
- le ridondanze e anomalie sono causate da dipendenze X → Y che causano ripetizioni all'interno della relazione (impiegato,stipendio e progetto, budget si ripetono nella relazione)

Forma normale di Boyce-Codd

Lo schema CSZ con dipendenze F={CS→ Z, Z→ C} non è in forma normale di Boyce-Codd

la dipendenza Z→ C viola la BCNF:

Z+ = {Z,C} e quindi non contenendo S, Z non è chiave

Forma normale di Boyce-Codd

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, poiché le uniche dipendenze funzionali sono quelle che hanno come lato sinistro la chiave

Terza forma normale

- Esiste una forma normale più debole della BCNF, nota come terza forma normale (3NF)
- Un attributo A di uno schema di relazione è un attributo primo se è elemento di una qualche chiave di R (nota: R può avere più chiavi)
- *Esempio*: CSZ con *F* = {CS→ Z, Z→ C}
  tutti gli attributi sono primi in quanto sia CS che SZ sono chiavi
- *Esempio*: ABCD con *F* = {AB→ C, B→ D, BC→ A}
  le chiavi sono AB e BC, quindi A,B,C sono primi, mentre D no

Terza forma normale

- Uno schema di relazione R con dipendenze *F* è in terza forma normale (3NF) se:
  - per ogni X→ A in F, tale che A non è in X, allora:
    (i) X è una superchiave di R (cioè X è la chiave o contiene la chiave)
    oppure
    (ii) A è primo
- La differenza fondamentale è che nella terza forma normale si ammettono dipendenze funzionali che non abbiano una chiave sul lato sinistro, purché sul lato destro si abbia un attributo primo

Esempio

- CSZ con *F* = {CS→ Z, Z→ C}
  - è in terza forma normale in quanto tutti gli attributi sono primi
- SAIP con *F* ={ SI→ P, S→ A}
  - l'unica chiave per questa relazione è 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 è superchiave e A non è primo

Esempio

- Schema R = (S, I, D, M)
  S = Store I=Item D=Department M=Manager
- Dipendenze funzionali:
  - SI→ D ogni prodotto è venduto da al più un dipartimento per negozio
  - SD→ M ogni dipartimento ha un solo manager
- la chiave è SI: si determina calcolando (SI)+
- la dipendenza SD → M viola la 3NF in quanto SD non è una superchiave per R e M non è primo

Scomposizione di schemi relazionali

- La scomposizione di uno schema relazionale R={A1, A2,...,An} è la sua sostituzione con una collezione C={R1, R2,...,Rk} di sottoinsiemi di R tali che R = R1 U R2 U... U Rk
- i vari schemi Ri non sono in genere disgiunti
- se si ha uno schema non normalizzato è possibile scomporlo in sottoschemi normalizzati

43

Scomposizione di schemi relazionali - esempi

- Una scomposizione (in BCNF) dello schema di esempio è
  impiegati (impiegato,stipendio)
  progetti (progetto,budget)
  lavora (impiegato,progetto,funzione)
- una scomposizione (in BCNF) di
  supplier (*sname, saddress, item, price)*
  con dipendenze funzionali
  sname → *saddress* e *sname item* → *price*
  è
  SA (*sname, saddress)* e *SIP(sname , item, price)*

44

Scomposizione di schemi relazionali

- Bisogna però che la scomposizione abbia alcune proprietà
- scomposizioni errate possono generare relazioni che, ricongiunte con il join, producono relazioni con dati incerti
- si ha quindi una perdita di informazione
- consideriamo un esempio di relazione:

  SEDI (impiegato, progetto, sede)
  con le dipendenze:

  impiegato → sede e
  progetto → sede

45

Scomposizione di schemi relazionali

chiave di SEDI:
impiegato, progetto

impiegato	progetto	sede
Rossi	biella	milano
Verdi	valvola	torino
Verdi	albero	torino
Bianchi	cinghia	milano
Neri	valvola	torino

decomponendo secondo le due dipendenze:

46

Scomposizione di schemi relazionali

impiegati

impiegato	sede
Rossi	milano
Verdi	torino
Bianchi	milano
Neri	torino

progetti

progetto	sede
biella	milano
valvola	torino
albero	torino
cinghia	milano

il join sull'attributo comune:

SELECT I.IMPIEGATO, P.PROGETTO, P.SEDE
FROM IMPIEGATI I, PROGETTI P
WHERE I.SEDE = P.SEDE

47

Scomposizione di schemi relazionali

impiegato	progetto	sede
Rossi	biella	milano
Rossi	cinghia	milano
Verdi	valvola	torino
Verdi	albero	torino
Bianchi	biella	milano
Bianchi	cinghia	milano
Neri	valvola	torino
Neri	albero	torino

crea tuple che non esistevano!

48

Scomposizioni lossless join

- Dato uno schema di relazione R, una scomposizione $R_1, R_2, \ldots R_k$ di R, e un insieme di dipendenze funzionali D, si dice che la scomposizione è lossless join (rispetto a D) se per ogni relazione r di schema R che soddisfa D si ha:

$$r = \Pi_{R_1}(r) \bowtie \ldots \bowtie \Pi_{R_k}(r)$$

cioè r è il join naturale delle sue proiezioni sui vari schemi R_i (i=1,..,k)
- se una relazione viene scomposta, è importante che sia possibile riottenerla eseguendo il join naturale delle relazioni in cui è stata scomposta

49

Scomposizioni che preservano le dipendenze

- Un'altra importante proprietà di una scomposizione di uno schema R in $\rho=(R_1,\ldots,R_k)$ è che l'insieme delle dipendenze *F* definito per R sia preservato dalle dipendenze che si hanno sugli schemi $R_1,\ldots,R_k$
- il motivo per cui si vogliono preservare le dipendenze è che tali dipendenze sono dei vincoli di integrità
- Dato *F* per un insieme di attributi D, e dato un insieme Z $\subseteq$ D, la proiezione di *F* su Z, denotata da $\Pi_Z(F)$, è l'insieme di dipendenze X→Y in *F+* tali che XY $\subseteq$ Z
  - Notare che X→Y non deve necessariamente essere in *F* è 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*

50

Scomposizioni che preservano le dipendenze

- Schema R=(CSZ)
  con dipendenze funzionali {CS→Z, Z→C}
  e scomposizione (lossless join) in R1=SZ e R2=CZ
- la proiezione di {CS→Z, Z→C} su SZ contiene solo le dipendenze banali che seguono dall'assioma di riflessività (cioè S→S e Z→Z)
- la proiezione su CZ contiene le dipendenze banali più la dipendenza Z→C
- la dipendenza CS→Z non è preservata in quanto le dipendenze banali e la dipendenza Z→C non implicano la dipendenza CS→Z

51

Scomposizioni che preservano le dipendenze

- Consideriamo la 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 ha due tuple che pur avendo lo stesso valore di C e S hanno valori diversi per Z

52

Scomposizioni lossless join e che preservano le dipendenze

- Esistono algoritmi (che non vediamo) che permettono di verificare se una scomposizione è lossless join
- Un modo per controllare se una scomposizione ρ =(R_1,......, R_k) preserva un insieme di dipendenze *F* è il seguente (che deriva dalla definizione):
  - si calcola *F+* e se ne esegue la proiezione su tutti gli R_i (i=1,...,k);
  - si esegue l'unione delle dipendenze ottenute come proiezione e si controlla se questo insieme implica tutte le dipendenze in *F*
- questo approccio è costoso perché il numero di dipendenze in *F+* è esponenziale nella dimensione di *F*
- esiste un algoritmo (che non vediamo) per verificare se una scomposizione preserva le dipendenze funzionali con costo polinomiale nella dimensione di *F*

53

Scomposizioni in BCNF e 3NF

- Qualsiasi schema di relazione ha una scomposizione BCNF che ha la proprietà di lossless join
- Qualsiasi schema di relazione ha un scomposizione 3NF che ha la proprietà di lossless join e preserva le dipendenze
- D'altra parte è possibile che non esista, per un dato schema di relazione, una scomposizione lossless join in BCNF 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
- Se non esiste scomposizione BCNF che preserva le dipendenze è preferibile usare la 3NF

54

Algoritmo per scomposizione lossless join in BCNF

- *Input*: Uno schema di relazione R e un insieme di dipendenze funzionali *F*
- *Output:* Una scomposizione di R con proprietà di lossless join, tale che ogni schema ottenuto dalla scomposizione è in BCNF rispetto alle proiezioni di *F* su tali schemi
- *Metodo:* Si costruisce iterativamente una scomposizione r per R

55

Algoritmo per scomposizione lossless join in BCNF

- Ad ogni passo ρ ha la proprietà di lossless join rispetto ad *F*
- Inizialmente, ρ contiene solo R
- (*) Se S è uno schema in ρ e S non è in BCNF,
  - sia $X \rightarrow A$ una dipendenza che vale per S, tale che X non è una superchiave di S e A non è 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 è in BCNF

56

Esempio

- R=CTHRSG
  C=course, T=teacher , H=hour, S = student G=grade R=room
- Valgono le seguenti dipendenze funzionali:
  - $C \rightarrow T$ ogni corso ha un solo insegnante
  - $HR \rightarrow C$ un solo corso può essere tenuto in una data aula ad una data ora
  - $HT \rightarrow R$ un docente non può essere sono in due aule contemporaneamente
  - $CS \rightarrow G$ ogni studente ha un solo voto per ogni esame
  - $HS \rightarrow R$ uno studente non può essere contemporaneamente in due aule
- Questo schema relazionale ha una sola chiave HS

57

Esempio

- consideriamo la dipendenza $CS \rightarrow G$ che viola la BCNF
- applicando l'algoritmo scomponiamo R in due schemi
  S1=(CSG) e S2=(CTHRS)
  nota: data una dipendenza $X \rightarrow A$, S1=X U A S2=R-A
- si calcolano le proiezioni di *F+* su CSG e CTHRS
  $\Pi_{CSG}(F) = \{CS \rightarrow G\}$ chiave CS
  $\Pi_{CTHRS}(F) = \{C \rightarrow T, HR \rightarrow C, TH \rightarrow R, HS \rightarrow R\}$ chiave HS
- CSG è in BCNF

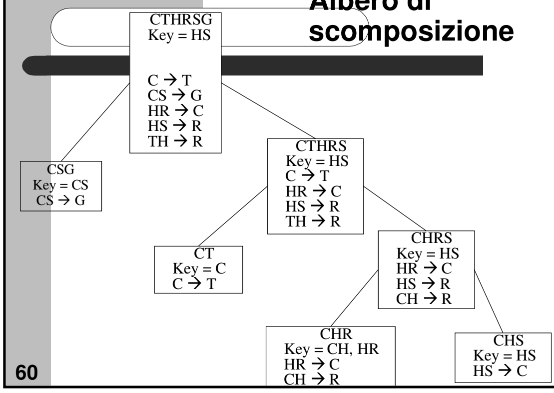
58

Esempio (cont.)

- CTHRS non è in BCNF
- consideriamo la dipendenza $C \rightarrow T$ che viola la BCNF e scomponiamo in CT e CHRS
  $\Pi_{CT}(F) = \{C \rightarrow T\}$ chiave C
  $\Pi_{CHRS}(F) = \{CH \rightarrow R, HS \rightarrow R, HR \rightarrow C\}$ chiave HS
  si noti che $CH \rightarrow R$ è necessaria nella proiezione su CHRS ma non in CTHRS in quanto in quest'ultima è implicata da $C \rightarrow T$ e $TH \rightarrow R$
- CT è in BCNF
- CHRS non è in BCNF
- consideriamo $CH \rightarrow R$ e scomponiamo in
  - CHR con $\{CH \rightarrow R, HR \rightarrow C\}$, con chiavi {CH, HR}
  - CHS con $\{HS \rightarrow C\}$, con chiave SH
- La scomposizione ottenuta è in BCNF

59

Albero di scomposizione



60

Esempio (cont.)

- La scomposizione finale è:
  - CSG: contiene il voto per studente per corso
  - CT: contiene il docente per ogni corso
  - CHR: contiene per ogni corso l'orario e l'aula
  - CHS: contiene per ogni studente i corsi e l'orario
- Si noti che si possono ottenere scomposizioni diverse a seconda delle dipendenze che si usano e in che ordine si esaminano
- Esempio: consideriamo l'ultimo passo
  - CHRS è stata scomposta in CHR e CHS essendo stata usata la dipendenza CH→R
  - usando invece HR→C si otterrebbe CHR e HRS

61

Esempio (cont.)

- La dipendenza TH→R non è preservata dalla scomposizione vista
- L'unione delle proiezioni di *F* su CSG, CT, CHR, e CHS è (in forma minimale) {CS→G, HR→C, C→T, CH→R,HS→C} e non implica 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

62

Algoritmo per scomposizione lossless join in 3NF

- *Input*: Uno schema di relazione R e un insieme minimale di dipendenze funzionali *F*
- *Output*: Una scomposizione lossless join di R tale che ogni schema è in 3NF rispetto ad *F* e ogni dipendenza è preservata
- *Metodo*: i=0; S=∅
  - **for each** dipendenza funzionale X→A in *F* **do begin**
    - i:=i+1;
    - R_i := XA;
    - S := S ∪ {R_i};

    end
  - **Optionally:** le relazioni di schema (XA_1),(XA_2),...,(XA_n) ottenute da dipendenze della forma X→A_1, X→A_2, ..., X→A_n possono essere sostituite dalla relazione di schema (X A_1 A_2 ... A_n)

63

Algoritmo per scomposizione lossless join in 3NF

- **if** nessuno schema R_k ($1 \le k \le i$) contiene una chiave candidata **then begin**
  - i:=i+1;
  - R_i:= una qualsiasi chiave candidata di R
  - S := S ∪ {R_i};

  end
- **for each** R_i ($R_i \in S$)
  - **if exists** $R_k \in S$, $i \ne k$, **such that** schema(R_k) ⊃ schema(R_i), **then** S:=S-{R_i};
- **return** (S)

64

Esempio:scomposizione in 3NF

- 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

65

Esempio

S = (J,K,L) con *F* = {JK→L, L→K} chiavi JK, JL

- Scomposizione in BCNF
  - S non è in BCNF
  - consideriamo la dipendenza L→K che viola la BCNF
  - otteniamo R1=(LK), R2=(JL)
  - questo schema non preserva la dipendenza JK→L
- Scomposizione in 3NF
  - lo schema S è già in 3NF in quanto
    - JK→L ha come lato sinistro una chiave
    - L→K ha come lato destro un attributo primo

66

Esempio

Schema:

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

- branch-name → assets
  branch-name → branch-city
  loan-number → amount
  loan-number → branch-name
- la chiave è (loan-number, customer-name)

67

Esempio

- Lo schema R non è in BCNF
  1) Consideriamo una dipendenza che viola le BCNF
  branch-name → assets
  - R è sostituita da
    - R1 = (branch-name, assets)
    - R2 =(branch-name, branch-city, loan-number, customer-name, amount)
  - R1 è in BCNF con chiave branch-name
  - R2 non lo è

68

Esempio

2) Consideriamo la dipendenza
branch-name→ branch-city
- R2 è sostituita da
  - R3 = (branch-name, branch-city)
  - R4 = (branch-name, loan-number, customer-name, amount)
- R3 è in BCNF con chiave branch-name
- R4 non lo è

69

Esempio

3) Consideriamo la dipendenza
loan-number→ amount
- R4 è sostuita da
  - R5 = (loan-number, amount)
  - R6 = (branch-name, loan-number, customer-name)
- R5 è in BCNF con chiave loan-number
- R6 non lo è

4) Consideriamo la dipendenza
loan-number → branch-name
- R6 è sostituita da
  - R7 = (loan-number, branch-name)
  - R8 = (loan-number, customer-name)

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

70

Utilizzo normalizzazione

- La progettazione di basi di dati basata sulla metodologia di progettazione proposta porta generalmente alla costruzione di schemi in 3NF
- può tuttavia capitare di
  - progettare in modo sbagliato lo schema concettuale
  - dovere eliminare anomalie da schemi non progettati secondo la metodologia proposta
  - volere uno schema in BCNF
- la normalizzazione può essere applicata sul risultato della progettazione logica, per scomporre le tabelle ottenute
  - semplice: le dipendenze coinvolgono singole tabelle

71