Esercitazione

Regole di inferenza

Regole di inferenza per le dipendenze funzionali:

(IR1) (Reflexive rule)
Se $X \supseteq Y$ allora $X \rightarrow Y$

(IR2) (Augmentation rule)
$X \rightarrow Y \models XZ \rightarrow YZ$

(IR3) (Transitive rule)
$\{X \rightarrow Y, Y \rightarrow Z\} \models X \rightarrow Z$

} **Regole di Armstrong**

(IR4) (Decomposition or Projective rule)
$\{X \rightarrow YZ\} \models X \rightarrow Z$

(IR5) (Union (or additive) rule)
$\{X \rightarrow Y, X \rightarrow Z\} \models X \rightarrow YZ$

(IR6) (Pseudotransitive rule)
$\{X \rightarrow Y, WY \rightarrow Z\} \models WX \rightarrow Z$

Proiezione
Dato un insieme di dipendenze F in R, la *proiezione* di F su R_i, denotata da π_{Ri} (F) (dove R_i è un sottoinsieme di R), è l'insieme di dipendenze $X \rightarrow Y$ in F^+ tale che gli attributi in $X \cup Y$ sono tutti contenuti in R_i.

Decomposizione con conservazione delle dipendenze
Diciamo che una decomposizione $D = \{R_1, R_2, \ldots, R_m\}$ di R è dependency-preserving rispetto a F se l'unione delle proiezioni di F su ciascun R_i in D è equivalente a F, cioè:

$$\{ (\pi R1 (F)) \cup \ldots \cup (\pi Rm (F)) \}^+ = F^+$$

Proprietà di lossless join

1. Creare una matrice S con una riga i per ogni relazione R_i nella decomposizione D, e una colonna j per ogni attributo A_i in R;
2. *porre* S(i,j) = b_{ij} per tutte le entrate della matrice;
   / ogni b_{ij} è un simbolo distinto associato agli indici (i,j) */*

3. *per ogni* riga i che rappresenta lo schema di relazione R_i
   per ogni colonna j che rappresenta l'attributo A_i
   se R_i include l'attributo A_i *allora porre* S(i,j) = a_i ;
4. *ripetere* quanto segue *finché* l'esecuzione del ciclo non modifica più S
   per ogni dipendenza funzionale $X \rightarrow Y$ in F
   per tutte le righe in S, che hanno gli stessi simboli nelle colonne corrispondenti agli attributi in X (a o b):
   rendere uguali i simboli in ogni colonna che corrispondono ad un attributo in Y, come segue:
   se ogni riga ha un simbolo **'a'** nella colonna, porre le altre righe allo stesso simbolo 'a' nella colonna.
   Se non esiste in nessuna riga un simbolo 'a' per l'attributo, scegliere uno dei simboli **'b'** che appaiono in una delle righe e porre le altre righe a quel simbolo 'b' nella colonna;
5. *se* una riga contiene solo simboli 'a', *allora* la decomposizione ha la proprietà lossless join, *altrimenti* no.

Decomposizione LJ e dependency-preserving in schemi di relazione in 3NF

Consente di ottenere in tre passi una decomposizione LJ e dependency-preserving con schemi di relazione in 3NF.

1. Trovare una copertura minimale G per F;
2. *per ogni* parte sinistra X di una FD in G, creare uno schema di relazione in D con attributi

   $\{X \cup A_1 \cup A_2 \cup \ldots \cup A_m\}$ dove $X \rightarrow A_1$, $X \rightarrow A_2$, …, $X \rightarrow A_m$ sono le sole dipendenze in G aventi X come parte sinistra.

3. *Se* nessuno degli schemi di relazione in D contiene una chiave di R, creare un altro schema di relazione che contiene attributi che formano una chiave per R.

Conservazione delle dipendenze

1) Let R{A,B,C,D} and
F={A→B, C→D }
Let's decomposed R into
R1(AB) R2(CD)

Is this decomposition dependency preserving? Yes.
F^+={A→B, C→D}
πR1(F)={A→B }
πR2(F)={C→D}
F^+={ πR1(F)$\cup$ πR2(F) }$^+$

1a) Let R{A,B,C,D} and
F={A→B, B→C, C→D, D→A}
Let's decomposed R into
R1(AB) R2(BC) R3(CD)

Is this decomposition dependency preserving? Yes.
F^+={A→B, B→C, A→C, C→D, A→D, B→D, D→A, B→A, C→A, C→B, D→B, D→C }
πR1(F)={A→B, B→A}
πR2(F)={B→C, C→B}
πR3(F)={C→D, D→C}

F^+={ πR1(F)$\cup$ πR2(F)$\cup$ πR3(F)}$^+$
A→B, B→C, C→D $\Longrightarrow$ A→D
B→A, C→B, D→C $\Longrightarrow$ D→A
…

---

2) Given the following: R(A,B,C,D,E)
F = {AB→C, C→E, B→D, E→A}
R1(B,C,D) R2(A,C,E)

Is this decomposition dependency preserving? **AB→C** is not preserved.
F^+={ AB→C, C→E, B→D, E→A, AB→E, C→A, AB→D }
πR1(F)={B→D }
πR2(F)={C→E, E→A, C→A}

---

3) Given R{A,B,C,D,E)
F={A→BD, B→E}
R1(A,B,C) R2(A,D) R3(B,D,E)

Is this a dependency preserving decomposition? Yes.
F^+={ A→BD, B→E, A→B, A→D, A→E }
πR1(F)={ A→B }
πR2(F)={ A→D }
πR3(F)={ B→E}

F^+={ πR1(F)$\cup$ πR2(F)$\cup$ πR3(F)}$^+$
A→B , A→D $\Longrightarrow$ A→BD
A→B , B→E $\Longrightarrow$ A→E

4) Given R{A, B, C, D, E, F}
F = { AC→B, FB→D, BD→C, F→ E, E→DF }
R1(ABC) R2(BCD) R3(CDE) R4(DEF) R5(ABF)

Is this a dependency preserving decomposition? Yes.
F^+={ AC→B, FB→D, BD→C, F→ E, E→DF, E→D, E→F, F→D, FD→C }
πR1(F)={ AC→B }
πR2(F)={ BD→C }
πR3(F)={ E→D }
πR4(F)={ F→ E,E→DF, E→D, E→F, F→D }
πR5(F)={ }

Concerning **FB→D:**

E→DF ⟹ E→D, E→F

F→ E, E→D ⟹ F→D ⟹ FB→DB

FB→DB ⟹ **FB→D**, FB→B

F^+={ πR1(F)∪ πR2(F)∪ πR3(F) ∪ πR4(F) ∪ πR5(F)}$^+$

Esercizio 11.29

Si considerino le decomposizioni seguenti per lo schema di relazione R dell'esercizio 10.26. Si determini se ogni decomposizione rispetto ad F gode:

(i) della proprietà di conservazione delle dipendenze,
(ii) della proprietà di join non-additivo.
(iii) Si determini anche in quale forma normale si trova ogni relazione della decomposizione.

a. **D_1 = { R_1, R_2, R_3, R_4, R_5 }**
R_1 = { A, B, C }, R_2 = { A, D, E }, R_3 = { B, F }, R_4 = { F, G, H }, R_5 = { D, I, J }

b. **D_2 = { R_1, R_2, R_3 }**
R_1 = { A, B, C, D, E }, R_2 = { B, F, G, H }, R_3 = { D, I, J }

c. **D_3 = { R_1, R_2, R_3, R_4, R_5 }**
R_1 = { A, B, C, D }, R_2 = { D, E }, R_3 = { B, F }, R_4 = { F, G, H }, R_5 = { D, I, J }

Ricordiamo che:
F = { { A, B } → { C }, { A } → { D, E }, { B } → { F }, { F } → { G, H }, { D } → { I, J } }

Risolviamo il punto a.

R_1 = { A, B, C }, R_2 = { A, D, E }, R_3 = { B, F }, R_4 = { F, G, H }, R_5 = { D, I, J }

(i) Questo set di relazioni ha la proprietà di conservazione delle dipendenze?

Per la proprietà di conservazione delle dipendenze, ogni dipendenza funzionale deve avere una relazione che la include. Perciò, questo insieme di relazioni ha questa proprietà.

(ii) Si applica l'algoritmo 11.1:

S	A	B	C	D	E	F	G	H	I	J
R_1	a1	a2	a3	~~b14~~ a4	~~b15~~ a5	~~b16~~ a6	~~b17~~ a7	~~b18~~ a8	~~b19~~ a9	~~b110~~ a10
R_2	a1	b22	b23	a4	a5	b26	b27	b28	~~b29~~	~~b210~~
R_3	b31	a2	b33	b34	b35	a6	~~b37~~ a7	~~b38~~ a8	b39	b310
R_4	b41	b42	b43	b44	b45	a6	a7	a8	b49	b410
R_5	b51	b52	b53	a4	b55	b56	b57	b58	a9	a10

Esiste una riga (R_1) composta da soli simboli "a", quindi la decomposizione ha la proprietà di lossless join.

(iii) In che forma normale sono R_1, …, R_5?

Questo set di relazioni è in 3NF.

Risolviamo il punto b.

R_1 = { A, B, C, D, E }, R_2 = { B, F, G, H }, R_3 = { D, I, J }

(i) Questo set di relazioni ha la proprietà di conservazione delle dipendenze?

Si, perché R_1 include le dipendenze funzionali **{ A, B } → { C }** e **{ A } → { D, E }**, R_2 include **{ B } → { F }** e **{ F } → { G, H }**, e R_3 include **{ D } → { I, J }**.

(ii) Si applica l'algoritmo 11.1:

S	A	B	C	D	E	F	G	H	I	J
R_1	a1	a2	a3	a4	a5	~~b16~~ a6	~~b17~~ a7	~~b18~~ a8	~~b19~~ a9	~~b110~~ a10
R_2	b21	a2	b23	b24	b25	a6	a7	a8	b29	b210
R_3	b31	b32	b33	a4	b35	b36	b37	b38	a9	a10

Esiste una riga (R_1) composta da soli simboli "a", quindi la decomposizione ha la proprietà di lossless join.

(iii) In che forma normale sono R_1, …, R_3?

R_1 è in 1NF, R_2 è in 2NF, e R_3 è in 3NF.

Risolviamo il punto **c.**
R_1 = { A, B, C, D }, R_2 = { D, E }, R_3 = { B, F }, R_4 = { F, G, H }, R_5 = { D, I, J }

(i) Questo set di relazioni ha la proprietà di conservazione delle dipendenze?

No, perché la dipendenza funzionale **{ A } → { D, E }** è inclusa in nessuna relazione.

(ii) Si applica l'algoritmo 11.1:

S	A	B	C	D	E	F	G	H	I	J
R_1	a1	a2	a3	a4	**b15**	~~b16~~ a6	~~b17~~ a7	~~b18~~ a8	~~b19~~ a9	~~b110~~ a10
R_2	b21	b22	b23	a4	a5	b26	b27	b28	~~b29~~ a9	~~b210~~ a10
R_3	b31	a2	b33	b34	b35	a6	~~b37~~ a7	~~b38~~ a8	b39	b310
R_4	b41	b42	b43	b44	b45	a6	a7	a8	b49	b410
R_5	b51	b52	b53	a4	b55	b56	b57	b58	a9	a10

Nessuna riga di S è composta da soli simboli "a", quindi la decomposizione **non** ha la proprietà di lossless join.

(iii) In che forma normale sono R_1, …, R_5?

R1 è in 1NF, R_2, …, R_5 sono in 3NF.

Esercizio

Dato il seguente schema di relazione R = (A, B, C, D, E) e il seguente insieme di dipendenze funzionali F = {C → D, AB → E, D → B} dire se la decomposizione r = {AC, ADE, CDE, AD, B} ha un join senza perdita.

Soluzione

Per trovare la chiave dello schema di relazione in base alle dipendenze date dobbiamo effettuare due passi:

1. identificare un insieme di attributi che attraverso le dipendenze funzionali date determina tutto lo schema
2. verificare che questo insieme sia minimale, cioè che non ammetta sottoinsiemi che abbiano la stessa proprietà di determinare tutto lo schema.

Analizzando le dipendenze, cominciamo col considerare gli insiemi di attributi che compaiono a sinistra delle dipendenze stesse e calcoliamone le chiusure utilizzando l'algoritmo studiato (se la chiusura di un insieme di attributi contiene l'intero schema l'insieme di attributi è ovviamente una superchiave):

$(AB)^+ = \{A,B,E\}$

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

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

La chiave è quindi (A,B,C).

È facile verificare che la chiave minimale (deve contenere necessariamente C) è AC.

Applichiamo l'algoritmo per verificare che una decomposizione ha un join senza perdita. Cominciamo a costruire la relativa tabella:

	A	B	C	D	E
AC	a1	b12	a3	b14→a4	b15
ADE	a1	b22→b12	b23	a4	a5
CDE	b31	b32→b12	a3	a4	a5
AD	a1	b42→b12	b43	a4	b45
B	b51	a2	b53	b54	b55

Per chiarezza applichiamo le dipendenze funzionali nell'ordine e vediamo i cambiamenti che vengono effettuati sulla tabella (ricordiamo che ogni cambiamento corrisponde a fare in modo che venga soddisfatta una dipendenza funzionale, per ottenere alla fine dell'algoritmo una tabella che rappresenta un'istanza legale dello schema):

C → D la prima e la terza riga coincidono sull'attributo C =a3, quindi cambiamo b14 in a4 in modo che la dipendenza funzionale sia soddisfatta (se le righe hanno valori uguali in C, devono avere valori uguali in D).

AB → E non viene utilizzata in questo passo: la dipendenza funzionale è già soddisfatta, in quanto non ci sono tuple uguali su AB e diverse su E, quindi non devono essere effettuati cambiamenti

D → B nelle prime quattro righe D = a4, quindi cambiamo b22 in b12, b32 in b12, b42 in b12 (potevano scegliere una diversa sostituzione delle b, purché le rendesse tutte uguali).

La tabella è diventata:

	A	B	C	D	E
AC	a1	b12	a3	a4	b15→a5
ADE	a1	b12	b23	a4	a5
CDE	b31	b12	a3	a4	a5
AD	a1	b12	b43	a4	b45→a5
B	b51	a2	b53	b54	b55

Poiché la tabella è cambiata, ripetiamo il ciclo e riapplichiamo le dipendenze:

C → D non viene utilizzata in questo passo: la dipendenza funzionale è già soddisfatta, in quanto non ci sono tuple uguali su C e diverse su D, quindi non devono essere effettuati cambiamenti

AB → E la prima, la seconda e la quarta riga coincidono sugli attributi AB = a1 b12, quindi cambiamo b15 in a5 e b45 in a5 in modo che la dipendenza funzionale sia soddisfatta (se le righe hanno valori uguali in AB, devono avere valori uguali in E)

D → B non viene utilizzata in questo passo: la dipendenza funzionale è già soddisfatta, in quanto non ci sono tuple uguali su D e diverse su B, quindi non devono essere effettuati cambiamenti

La tabella è diventata

	A	B	C	D	E
AC	a1	b12	a3	a4	a5
ADE	a1	b12	b23	a4	a5
CDE	b31	b12	a3	a4	a5
AD	a1	b12	b43	a4	a5
B	b51	a2	b53	b54	b55

Poiché la tabella è cambiata, ripetiamo il ciclo e riapplichiamo le dipendenze:

C → D non viene utilizzata in questo passo

AB → E non viene utilizzata in questo passo

D → B non viene utilizzata in questo passo

La tabella non cambia più e l'algoritmo termina. Non essendoci una riga con tutte a, il join NON È senza perdita.

Esercizio

Dato il seguente schema relazionale: R = (A, B, C, D, E, F) con associato l'insieme di dipendenze funzionali: F = { AC → B, FB → D, BD → C, F → E, E → DF }
Stabilire se la decomposizione S = (ABC, BCD, CDE, DEF, ABF) è o meno una decomposizione lossless join che conserva le dipendenze.

Si applica l'algoritmo 11.1:

S	A	B	C	D	E	F
R_1	a1	a2	a3	b14	b15	b16
R_2	b21	a2	a3	a4	b25	b26
R_3	b31	b32	a3	a4	a5	b36
R_4	b41	b42	b43	a4	a5	a6
R_5	a1	a2	b53	b54	b55	a6

Si applicano in sequenza AC → B, FB → D F → E, E → DF ed infine BD → C

S	A	B	C	D	E	F
R_1	a1	a2	a3	~~b14~~ a4	b15	b16
R_2	b21	a2	a3	a4	b25	b26
R_3	b31	~~b32~~ a2	a3	a4	a5	~~b36~~ a6
R_4	b41	b42	b43	a4	a5	a6
R_5	a1	a2	~~b53~~ a3	~~b54~~ a4	~~b55~~ a5	a6

Esercizio

Dato lo schema di relazione **R = (a, b, c, d, e, f, g, h, i)** con chiave **abcd** e le seguenti dipendenze funzionali:
a → ei **bd → h** **c → f**

trovare una decomp. di R in schemi 3NF che preservi le dipendenze e dia un join senza perdite.

Soluzione

Poiché la chiave **abcd** della relazione **R = (a, b, c, d, e, f, g, h, i)** è data, alle dipendenze date **a → ei**, **bd → h**, **c → f** vanno aggiunte quelle determinate dal vincolo di chiave, cioè

abcd → e,
abcd → f,
abcd → g,
abcd → h,
abcd → i

perché potrebbero essere non ridondanti (infatti vediamo subito che **g** non è determinato da nessun attributo o insieme di attributi nelle dipendenze date, ma ovviamente sarà determinato per definizione dalla chiave). L'insieme iniziale di dipendenze è quindi:
FO = { abcd → e, abcd → f, abcd → g, abcd → h, abcd → i, a → ei, bd → h, c → f }

Abbiamo quindi le dipendenze parziali **a → ei** (che per decomposizione si spezza in **a → e** e **a → i**), **bd → h**, **c → f**, quindi **R** non è in 3NF.

Passiamo a determinare una copertura minimale per FO, che dopo il primo passo dell'algoritmo, che applica la regola di decomposizione per ridurre le parti destre a singleton, è:
F={ abcd → e, abcd → f, abcd → g, abcd → h, abcd → i, **a → e**, **a → i**, bd → h, c → f }

Passando al secondo passo, cominciamo a provare a ridurre **abcd → e** .
Possiamo già prevedere di poterla sostituire con **a → e** (e quindi eliminarla in quanto duplicato) perché **a ⊆ abcd** , ma a scopo didattico e solo per questa dipendenza seguiamo il procedimento che ci porterebbe alla stessa conclusione.

Iniziamo provando a sostituire **abcd → e** con **bcd → e**. Dobbiamo verificare se **e ∈ (bcd)$^{+F}$**, quindi applichiamo il relativo algoritmo.
All'inizio **Z := bcd**, **S := hf** grazie alle dipendenze **bd → h** con **bd ⊆ Z** e **c → f** con **c ⊆ Z**;
alla prima iterazione del ciclo while abbiamo **Z := bcdhf**, **S := hf**, in quanto l'aggiunta a **Z** di **h** ed **f** non ci permette di considerare nuove dipendenze; poiché **S ⊆ Z**, l'algoritmo si ferma con **Z = bcdhf**.
L'attributo **e** non fa parte di questo insieme, quindi **abcd → e** non può essere sostituita con **bcd → e**. In ogni caso il calcolo fatto di **(bcd)$^{+F}$ = bcdhf** ci servirà di nuovo in seguito.
Proviamo allora a sostituire **abcd → e** con **acd → e**.
Applicando l'algoritmo, all'inizio **Z := acd**, **S := eif**, grazie alle dipendenze **a → e**, **a → i**, **c → f**;
alla prima iterazione del while abbiamo **Z := acdeif**, **S := eif** perché non possiamo considerare nuove dipendenze; poiché **S ⊆ Z** l'algoritmo si ferma con **Z = acdeif**, che contiene l'attributo **e**, quindi la sostituzione può avvenire. Ricordiamo comunque il risultato **(acd)$^{+F}$ = acdeif**.

A questo punto:
F = { acd → e, abcd → f, abcd → g, abcd → h, abcd → i, a → e, a → i, bd → h, c → f }

Proviamo a ridurre ancora **acd → e**.

Provare con **cd → e** è inutile, in quanto essendo **cd ⊆ bcd**, avremo sicuramente $(cd)^{+F} \subseteq (bcd)^{+F}$, e avevamo già verificato che **e ∉ (bcd)$^{+F}$**. Proviamo allora con **ad → e**, verificando se **e ∈ (ad)$^{+F}$**.

All'inizio dell'algoritmo **Z := ad**, **S := ei** grazie alle dipendenze **a → e** ed **a → i**; alla prima iterazione del while abbiamo **Z := adei**, **S := ei ⊆ Z** quindi l'algoritmo termina con **Z =(ad)$^{+F}$ = adei**, che contiene l'attributo **e**; la sostituzione può essere effettuata, quindi

F={ ad → e, abcd → f, abcd → g, abcd → h, abcd → i, a → e, a → i, bd → h, c → f}

Provando ulteriormente a ridurre **ad → e**, vediamo che **a → e** appartiene già ad F, quindi **ad → e** può essere eliminata del tutto. Era questo il risultato che avevamo già previsto. Abbiamo allora:

F={ abcd → f, abcd → g, abcd → h, abcd → i, a → e, a → i, bd → h, c → f }

Passiamo alla dipendenza **abcd → f**. La eliminiamo senza ulteriori verifiche in quanto, dopo una sequenza di eliminazioni di attributi a sinistra, potrà essere sicuramente sostituita da **c → f** (essendo **c ⊆ abcd**) e quindi eliminata come duplicato. Quindi ora:

F={ abcd → g, abcd → h, abcd → i, a → e, a → i, bd → h, c → f }

Prendiamo in considerazione **abcd → g**, e osserviamo che non esiste in F una dipendenza **Y→ g** con **Y ≠ abcd**. Di conseguenza è inutile provare a ridurre questa dipendenza.

Esaminiamo allora **abcd → h**. Questa dipendenza può essere eliminata osservando che in **F** abbiamo **bd → h** (**bd ⊆ abcd**), che sostituirebbe **abcd → h** dopo una sequenza di eliminazioni di attributi a sinistra, portando a un duplicato.

Quindi F={abcd → g, abcd → i, a → e, a → i, bd → h, c→ f }

Analogo discorso vale per la dipendenza **abcd → i**, grazie alla presenza di **a → i**.

Ora F={abcd → g, a → e, a → i, bd → h, c→ f }

Rimane da verificare se si può ulteriormente ridurre **bd → h**.

Tuttavia, anche senza effettuare i calcoli, per la forma delle dipendenze di **F**, è facile convincersi che **(b)$^{+F}$ = b** e **(d)$^{+F}$ = d** (applicando l'algoritmo per la chiusura ai due attributi separatamente, abbiamo in entrambi i casi un insieme S vuoto).

All'insieme F={abcd $\rightarrow$ g, a $\rightarrow$ e, a $\rightarrow$ i, bd $\rightarrow$ h, c$\rightarrow$ f } dovremmo ora applicare il terzo passo per la copertura minimale.

Tuttavia vediamo che tutti gli attributi delle parti destre compaiono in esattamente una dipendenza, quindi non troveremmo dipendenze ridondanti.

Infine applichiamo l'algoritmo per la decomposizione di R in base al nuovo insieme **F**. Non abbiamo attributi che non partecipano ad alcuna dipendenza, né dipendenze che coinvolgono tutti gli attributi, quindi procediamo secondo le dipendenze presenti in F ottenendo la decomposizione:

D = {abcdg, ae, ai, bdh, cf}

La chiave originaria **abcd** è gia presente in uno degli elementi di D, per cui la decomposizione cercata è

R1 = {abcdg}, R2 = {aei}, R3 = {bdh}, R4 = {cf}.

Algoritmo di sintesi relazionale in 3NF con conservazione delle dipendenze e lossless join

Usiamo l'algoritmo 11.4 per trovare la collezione di relazioni in 3NF con conservazione delle dipendenze le quali hanno la proprietà di lossless join.

Date le dipendenze funzionali (esercizio 10.26):
F:

1. AB → C
2. A → DE
3. B → F
4. F → GH
5. D → IJ

F^+ (chiusura delle parti sinistre):

$\{ A, B \}^+ = \{ A, B, C, D, E, F, G, H, I, J \}$
$\{ A \}^+ = \{ A, D, E, I, J \}$
$\{ B \}^+ = \{ B, F, G, H \}$
$\{ F \}^+ = \{ F, G, H \}$
$\{ D \}^+ = \{ D, I, J \}$

{ A, B } è la chiave di R.

Passo 1.
Copertura Minima (algoritmo 10.2): Si scrivano le dipendenze funzionali con i RHS singoli. Verifica la presenza di attributi ridondanti sulla relazione **{ A, B } → { C }**: non è possibile sostituirla con **{ A } → { C }** o **{ B } → { C }** (basta fare la chiusura della LHS rispetto le DF senza considerare **{ A, B } → { C }** e quindi verificare la presenza o no della RHS).
(LHS = Left-Hand-Side e RHS = Right-Hand-Side)

Verificare la presenza di dipendenze funzionali ridondanti (non ce ne sono).

1: AB → C
6: A → D [decomposizione di 2]
7: A → E [decomposizione di 2]
3: B → F
8: F → G [decomposizione di 4]
9: F → H [decomposizione di 4]
10: D → I [decomposizione di 5]
11: D → J [decomposizione di 5]

Passo 2. Creare una relazione per ogni dipendenza funzionale ed unire quelle che hanno lo stesso LHS; quindi le relazioni create sono:

R_1 = { A, B, C }, R_2 = { A, D, E }, R_3 = { B, F }, R_4 = { F, G, H }, R_5 = { D, I, J }

Passo 3. È stato completato poiché R_1 contiene la chiave primaria **{ A, B }**.

Esercizio

Dato il seguente schema di relazione **R = (A, B, C, D, E, H)** e il seguente insieme di dipendenze funzionali F = { AB $\rightarrow$ CD, C $\rightarrow$ E, AB $\rightarrow$ E, ABC $\rightarrow$ D }

a) Verificare che ABH è una chiave per R.
b) Sapendo che ABH è l'unica chiave per R, verificare che R non è in 3NF.
c) Trovare una copertura minimale G di F.
d) Trovare una decomposizione r di R tale che preserva G e ogni schema in r è in 3NF.
e) Trovare una decomposizione s di R tale che preserva G, ha un join senza perdita e ogni schema in s è in 3NF.

Soluzione

a) Verificare che ABH è una chiave significa verificare due condizioni

1. ABH deve determinare funzionalmente l'intero schema.
2. Nessun sottoinsieme di ABH deve determinare funzionalmente l'intero schema.

Per verificare la prima condizione si controlla se la chiusura dell'insieme di attributi ABH contiene tutti gli attributi dello schema.

Applicando l'algoritmo per il calcolo della chiusura di un insieme di attributi, al primo passo inseriamo nella chiusura di ABH gli attributi C, D ed E grazie alle dipendenze funzionali in cui la parte sinistra e' contenuta in ABH, cioè AB $\rightarrow$ CD e AB $\rightarrow$ E, quindi possiamo anche fermarci perché abbiamo inserito tutti gli attributi dello schema, cioè abbiamo verificato $(ABH)^{+}$ = {A,B,C,D,E,H}.

Per verificare la seconda condizione, dobbiamo controllare che la chiusura di nessun sottoinsieme di ABH contenga tutti gli attributi dello schema.

A questo proposito notiamo che H deve comparire in ogni caso in una chiave dello schema, perché non viene determinato da altri attributi. Quindi possiamo evitare di calcolare le chiusure degli attributi singoli A e B che non potrebbero determinare H neppure per transitività. Inoltre H non compare mai a sinistra delle dipendenze, quindi da solo non determina alcun attributo.

Restano da controllare le chiusure di AH e di BH, ma è banale verificare che in entrambi i casi l'algoritmo terminerebbe subito perché non ci sono dipendenze con a sinistra sottoinsiemi di AH o di BH.

Possiamo quindi concludere che ABH è chiave dello schema dato.

b) Per verificare che lo schema non è in terza forma normale, basta osservare la presenza delle dipendenze parziali AB $\rightarrow$ CD e AB $\rightarrow$ E.

c) Per trovare la copertura minimale, prima di tutto riduciamo le parti destre a singleton

F = {AB $\rightarrow$ C, AB $\rightarrow$ D ,C $\rightarrow$ E, AB $\rightarrow$ E, ABC $\rightarrow$ D}

Ora dobbiamo verificare se nelle dipendenze ci sono ridondanze nelle parti sinistre. Cominciamo dalla dipendenza AB $\rightarrow$ C e controlliamo se A $\rightarrow$ C appartiene a F^{+}, cioè se C appartiene a (A)+F .

Provando ad applicare l'algoritmo non potremmo inserire nessun attributo nella chiusura di A, in quanto non ci sono dipendenze che abbiano nella parte sinistra il solo attributo A, quindi $(A)^{+F} = \{A\}$.

Lo stesso si verifica per B, cioè $(B)^{+F} = \{B\}$, quindi la parte sinistra della dipendenza non può essere ridotta.

Lo stesso si verifica per le dipendenze $AB \rightarrow D$ e $AB \rightarrow E$.

Proviamo allora a ridurre $ABC \rightarrow D$. Poiché nell'insieme di dipendenze esiste $AB \rightarrow D$, possiamo non solo eliminare l'attributo C ma anche tutta la dipendenza risultante che è un duplicato.

Alla fine di questo passo abbiamo un insieme $G = \{ AB \rightarrow C, AB \rightarrow D, C \rightarrow E, AB \rightarrow E\}$ di dipendenze equivalente ad F.

Vediamo ora se questo insieme contiene delle dipendenze ridondanti.

Intanto possiamo considerare che C viene determinato unicamente da AB, quindi eliminando la dipendenza $AB \rightarrow C$ non riusciremmo più ad inserirlo nella chiusura di AB rispetto al nuovo insieme di dipendenze. Lo stesso vale per D.

Proviamo allora ad eliminare la dipendenza $C \rightarrow E$. Rispetto al nuovo insieme di dipendenze $G = \{ AB \rightarrow C, AB \rightarrow D, AB \rightarrow E\}$ abbiamo che $(C)^{+G} = \{C\}$ in cui non compare E. La dipendenza dunque deve rimanere.

Proviamo infine ad eliminare $AB \rightarrow E$. Rispetto a $G = \{ AB \rightarrow C, AB \rightarrow D, C \rightarrow E\}$ abbiamo che $(AB)^{+G} = \{A,B,C,D,E\}$ in cui E viene aggiunto al secondo passo dell'algoritmo per il calcolo della chiusura. Ciò significa che E rientra comunque nella chiusura di AB perché la dipendenza $AB \rightarrow E$, pur non comparendo in G, si trova in G^+, e quindi può essere eliminata. La copertura minimale di F sarà allora $G = \{AB \rightarrow C, AB \rightarrow D, C \rightarrow E\}$.

d) Applichiamo l'algoritmo per la decomposizione dello schema di relazione che non è 3NF dato l' insieme di dipendenze G che è una copertura minimale.

Al primo passo dobbiamo inserire in un elemento della decomposizione gli attributi che non compaiono nelle dipendenze di G. È il caso dell'attributo H, quindi inizialmente avremo $r = \{H\}$ e lo schema R diventa (A,B,C,D,E).

Passiamo poi a verificare che non ci sono dipendenze che coinvolgono tutti gli attributi dello schema, per cui eseguiamo il passo alternativo. Abbiamo alla fine $r = \{H, ABC, ABD, CE\}$.

e) Per avere una decomposizione con join senza perdita, aggiungiamo alla decomposizione precedente un sottoschema che contenga la chiave ABH (che non è già contenuta in alcuno degli schemi ottenuti). Avremo quindi

S = {{H}, {A,B,C,D}, {C,E}, {A,B,H}}

Esempio di decomposizione in 3NF

Consider the following example: R(A,B,C,D,E,F,G)
F= { AB→CD, C→EF, G→A, G→F, CE→F }

Step 1:

- Find the attributes that could be keys for the relation since no primary key is identified.
- G must be in any key since it is not determined by anything (does not appear in the right side of any dependency)
- B must be in any key for the same reason. Since $\{GB\}^+$ contains all the others (i.e. GB determine everything else by transitivity) GB is a key.
- Since any key must contain GB, GB is in fact the only minimal key.

Step 2:

- Clean up the set of dependencies by removing any that are implied by others (duplicates), or have unnecessary attributes in their left hand sides: CE→F is removed due to the presence of C→F.
- You can also combine dependencies with the same left hand sides. So F will reduce as follows.
  combine G→F and G→A into G→AF
  combine C→E and C→F into C→EF
- So we have F'_c = { AB→CD, C→EF, G→AF }

Step 3:

- Make a table for each of these dependencies with the left hand of the dependency being a key.
  R1=<u>AB</u>CD, R2=<u>C</u>EF, R3=<u>G</u>AF.
- If none of these tables contains the original key (GB) make a table for the key. So we have as our decomposition:
  R1=<u>AB</u>CD, R2=<u>C</u>EF, R3=<u>G</u>AF, R4=<u>GB</u>.

Esempio di decomposizione lossless, dependency preserving in 3NF

Consider the following example: R(A,B,C,D,E,F)

F ={A→BCE, B→DE, ABD→CF, AC→DE}

The key is A: In fact $\{A\}^+$={A, B, C, E, D, F}

Find a canonical cover for F:

A→B, A→C, A→E

B→D, B→E

ABD→C, ABD→F

AC→D, AC→E

Rule ABD→C becomes A→C

Rule ABD→F becomes A→F

In fact {X → Y, WY → Z} ⊨ WX → Z thus A→B and ADB→F ⊨ ADA→F = AD→F

Moreover: A→B, B→D ⊨ A→D and AD→F ⊨ A→F

Rule AC→D becomes A→D

Rule AC→E becomes A→E

The result is:

A→B, A→C, A→E

A→F

B→D, B→E

A→D

Removing Redundant Rule:

A→E is redundant: A→B and B→E

A→D is redundant: A→B and B→D

A→B, A→C

A→F

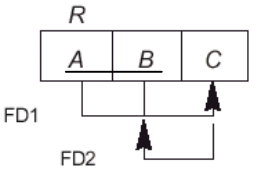
B→D, B→E

Finally: F= { A→BCF, B→DE }

Final Partition D = (<u>A</u>BCF, <u>B</u>DE)

One of these tables contains the original key A.

BCNF
Uno schema R è in BCNF se ogniqualvolta vale in R una dipendenza funzionale X → A, allora X è una superchiave di R.



è in 3NF ma non in BCNF

Decomposizione LJ in schemi in BCNF
Consente di ottenere in due passi una decomposizione LJ con schemi in BCNF:
porre D := R;
finché ci sta uno schema di relazione Q in D che non è in BCNF
{
si scelga uno schema di relazione Q in D che non sia in BCNF;
si trovi una dipendenza funzionale X → Y che violi BCNF;
si sostituisca Q in D con due schemi di relazione (Q-Y) e (X ∪ Y).
}

Esempio di decomposizione lossless, dependency preserving in BCNF
Suppose we have a database for an investment firm, consisting of the following attributes:
B – Broker, O – Office of a broker, I – Investor, S – Stock
Q – Quantity of stock owned by an investor, D – dividend paid by a stock.
Hence, the overall schema is R = (B, O, I, S, Q, D). Assume that the following f.d. are required to hold on this db: I → B, IS → Q, B → O, S → D.

1) List all the candidate keys for R.
2) Give a lossless-join decomposition of R into BCNF.
3) Give a lossless-join decomposition of R into 3NF preserving f.d. Is you answer is in BCNF?

1) I and S must be in any candidate key since they do not appear on the right of any f.d. The question is whether they form a complete candidate key. And yes, IS → ISDBOQ. Hence, the only candidate key is IS.

2)
1. Decompose R by I → B into R1 = (I, B), R2 = (I, O, S, Q, D).
2. R1 is in BCNF.
3. Decompose R2 by S → D into R21 = (S, D), R22 = (O, I, S, Q).
4. R21 is in BCNF.
5. Decompose R22 by I → O (I → B, B → O) into R221 = (I, O), R222 = (I, S, Q).
6. R221 is in BCNF.
7. R222 is in BCNF.

The decomposition is (I, B), (S, D), (I, O), (I, S, Q).

An alternative answer is (I, B), (S, D), (B, O), (I, S, Q), starting from B → O.

3) (I, B), (S, D), (B, O), (I, S, Q).
The answer is in BCNF.