UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

BASI DI DATI II

Progettare basi di dati di qualità: **equivalenza di insiemi di FD, Forme Normali, Algoritmi di Progettazione di Database relazionali**

Anno 2024/2025
Prof. Genny Tortora
Dott. Luigi Di Biasi

Esercitazione

G copre F?

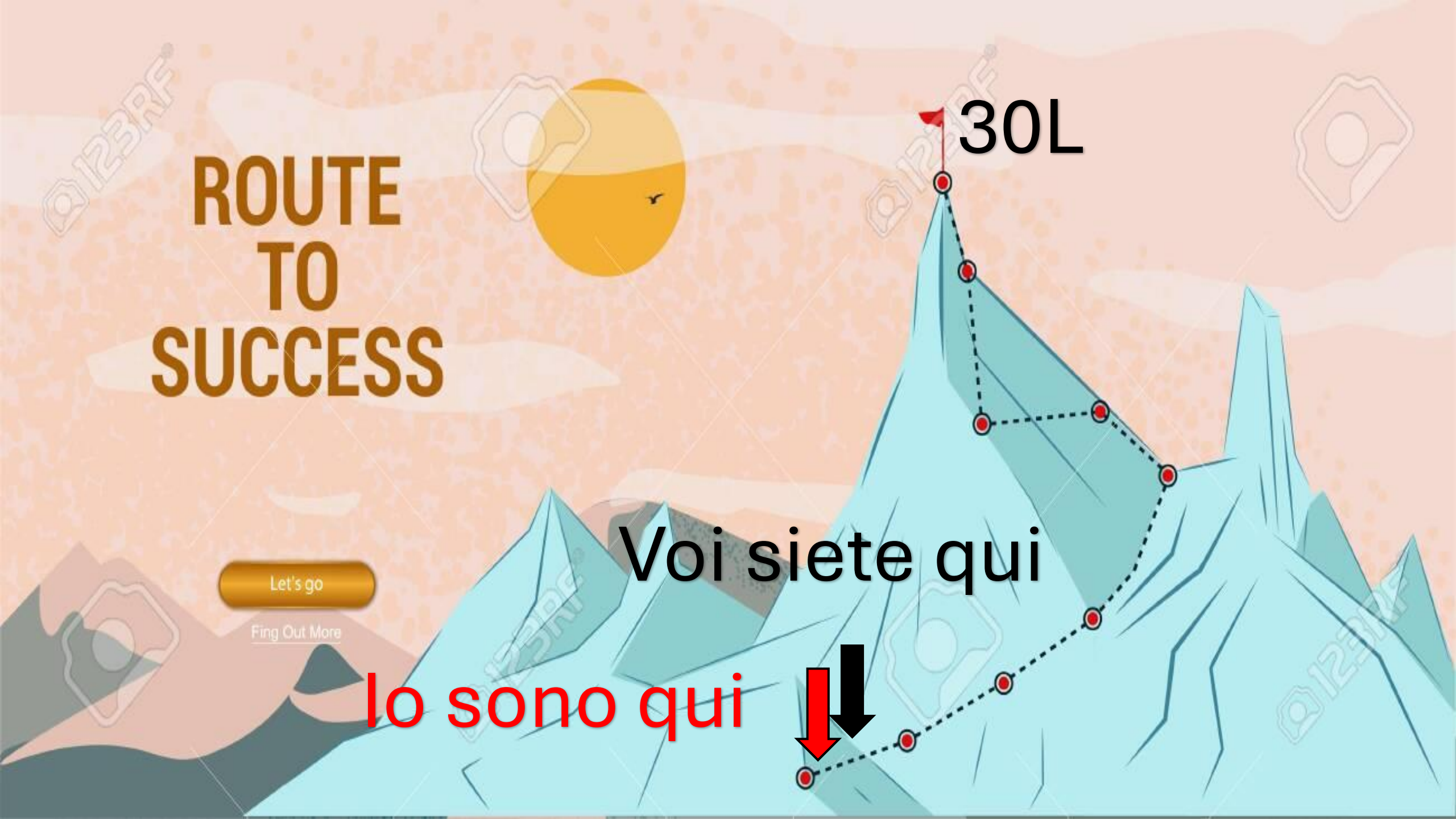
F = {
B → C,
B → A,
AD → E,
BD → F
}

G = {
B → C,
B → A,
AD → E,
AD → F

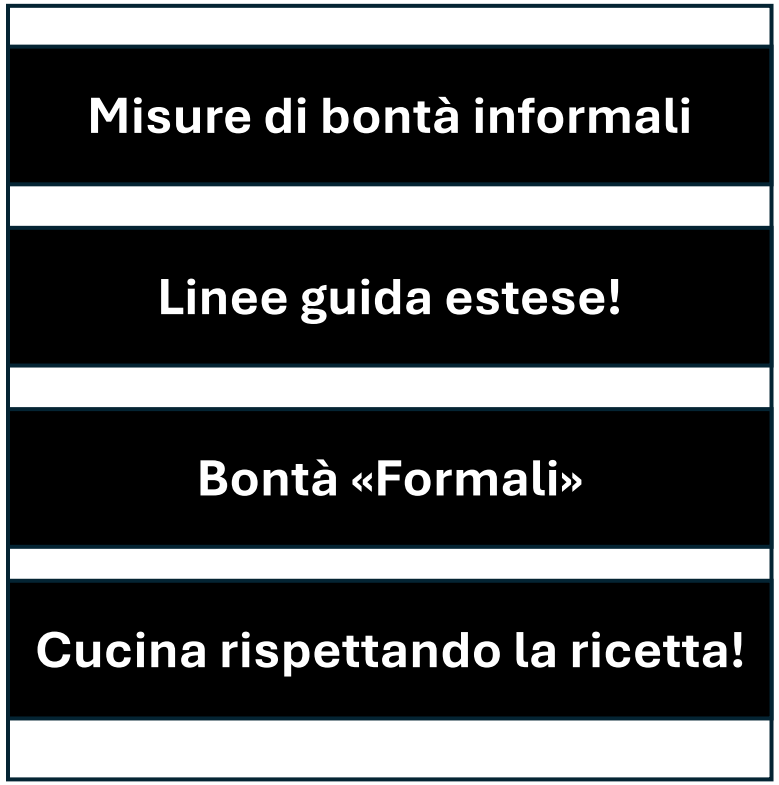
ERRORE?

$\{B\}^{+G} = \{B,C,A\}$ ⇔ B→ C e B→ A in F sono coperti da G.
$\{AD\}^{+G} = \{A,D,E,F\}$ ⇔ AD → E e AD→ F per la regola di decomposizione.

Non è possibile usare la regola transitiva B→ A dunque BD→F.


ROUTE
TO
SUCCESS
Let's go
Fing Out More
30L
Voi siete qui
Io sono qui

La ricetta della buona decomposizione



Come se non bastassero già:

- Le buone norme di progettazione (le consuetudini);
- Le linee guida estese (un po' più restrittive e formali);
- Le regole formali di BD1 (ristruttura e normalizza!)

il dio dei database relazionali, quando lavoriamo nel mondo del lavoro «reale» **ci impone di seguire una ricetta specifica affinché il database venga «cucinato» bene.**

Due diversi approcci alla progettazione

Approccio Top-Down (usato a Basi di Dati 1)

Si parte dal diagramma ER (o EER), si mappa in uno schema relazionale e si applicano le procedure di normalizzazione.

Approccio Bottom-Up (quello che userete a BD2)

È più formale, poiché considera lo schema del db solo in termini di dipendenze funzionali. **Dopo aver definito tutte le FD**, si applica un algoritmo di normalizzazione per sintetizzare gli schemi di relazione in 3NF o in BCNF (o in 4NF e 5NF!)

Due diversi approcci alla progettazione

Approccio Top-Down (usato a Basi di Dati 1)

Si parte dal diagramma ER (o EER), si mappa in uno schema relazionale e si applicano le procedure di normalizzazione.

Approccio Bottom-Up (quello che userete a BD2)

È più formale, poiché considera lo schema del db solo in termini di dipendenze funzionali. **Dopo aver definito tutte le FD**, si applica un algoritmo di normalizzazione per sintetizzare gli schemi di relazione in 3NF o in BCNF (o in 4NF e 5NF!)

Nota importante per la prova scritta!

Durante il corso di BD2 **consideriamo implicito usare l'approccio Bottom up.**

Di conseguenza, eventuali domande allo scritto del tipo ***«Decomporre in 3NF il seguente schema di relazione R relativo all'insieme di dipendenze funzionali FD»*** vanno intese come:

«Utilizzare gli algoritmi visti a BD2 – BOTTOM UP- per decomporre lo schema R garantendo la dependency-preserving!»

Approccio Bottom up

Durante il corso di BD2, useremo sempre questo tipo di approccio. Di conseguenza, **avremo sempre a disposizione**:

- Uno schema di relazione globale R = {A1, … An} ;
- Un insieme di dipendenze funzionali F = { → }

Studieremo quindi **degli algoritmi** che a partire da uno schema di relazione R globale e da un insieme di dipendenze funzionali (dipendenze semantiche tra gli attributi), ci permetteranno di avere in ouput una decomposizione di R **tale che soddisfi alcune proprietà di integrità** di interesse.

Formalmente che significa decomporre?

Gli algoritmi che presentiamo **partono da un singolo schema di relazione universale** $R = \{A_1, A_2, \ldots, A_n\}$ **che include tutti gli attributi** del database.

I progettisti specificano l'insieme F delle dipendenze funzionali che devono valere sugli attributi di R.

Usando le dipendenze funzionali, gli algoritmi decompongono lo schema di relazione R in un insieme di schemi $D = \{R_1, R_2, \ldots, R_n\}$ in cui D è detto «decomposizione di R»

Durante il corso di BD1 l'insieme di FD non lo avevate! Da qui dovreste rendervi conto che l'algoritmo da usare è diverso.

Legge 1: «Conserva degli attributi!»

Dato uno schema universale R = {A_1, A_2, …,A_n} **che include tutti gli attributi** del database e data una decomposizione D = {R1, … Rn}, affinché tale decomposizione è necessario che la stessa **CONSERVI TUTTI GLI ATTRIBUTI (chiaramente, non vogliamo perdere dati).**

$$\bigcup_{i=1}^{n} R_i = R$$

Se vi accorgente che l'unione delle decomposizioni non copre tutti gli attributi iniziali di R, avete sbagliato qualcosa.

Gli algoritmi che vedremo dovranno garantirci la conservazione degli attributi.

Legge 2: «rendi ogni R_i in D in BCNF (o in 3NF)!»

Altro obiettivo è che ogni relazione individuale Ri in D **deve essere in BCNF (o in 3NF).** Da notare che qualsiasi schema di relazione con soli due attributi è automaticamente in BCNF.

Questa condizione non è sufficiente per garantire di avere un buon db, **in quanto il fenomeno delle tuple spurie può comunque presentarsi anche con relazioni in BCNF.**

Quindi? Perché lo facciamo?

Legge 2: «rendi ogni R_i in D in BCNF (o in 3NF)!»

Una decomposizione in BCNF o 3NF ci aiuta ad evitare problemi di ridondanza che potrebbero causare inconsistenze nei dati. **Inoltre, aiuta a mantenere la coerenza delle dipendenze funzionali.**

A livello di performance, tabelle più piccole sono più efficienti da gestire rispetto a tabelle con molteplici attributi ridondanti.

Infine, ci garantisce una migliore gestione delle dipendenze funzionali (Assicurandoci che non ci siano dipendenze parziali o transitive indesiderate).

Legge 3: «non perderti dipendenze funzionali!»

Abbiamo detto che le dipendenze funzionali **sono vincoli semantici tra gli attributi**.

Decomporre uno schema R in D e perdere qualche dipendenza funzionale implicherebbe perdere in modo definitivo informazioni circa la correlazione semantica tra insiemi di attributi (ovvero, potremmo non riuscire più a capire come sono collegati i dati).

Dovrebbe apparire evidente che **è fondamentale non perdere delle dipendenze**, poiché queste rappresentano dei vincoli sul database.

Dunque, ci aspettiamo che gli algoritmi che studieremo ci aiuteranno a rispettare la 3° Legge.

Legge 3: «non perderti dipendenze funzionali!»

Alcuni chiarimenti importanti.

1. **Non è necessario che esattamente le stesse dipendenze specificate in F appaiono nelle relazioni della decomposizione D.**

2. **è sufficiente che l'unione delle dipendenze che valgono nelle singole relazioni di D sia equivalente ad F (devono avere lo stesso potere espressivo in D)**

Legge 3: «non perderti dipendenze funzionali!»

Alcuni chiarimenti importanti.

Sarebbe utile che le dipendenze funzionali X → Y specificate in F, apparissero direttamente in una delle R_i della decomposizione D **o possano essere inferite da dipendenze in altre R_i.**

Questa è (informalmente) la condizione di conservazione delle dipendenze.

Ovvero, dopo una decomposizione, mantengo intatto il link semantico tra i dati.

Legge 3: «non perderti dipendenze funzionali!»

Come formalizziamo la proprietà di «dependency preserving»?

Per farlo abbiamo bisogno di alcune definizioni.

Definizione:

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

In altre parole, la proiezione di F su Ri è l'insieme di tutte le regole di F (e delle loro conseguenze) **che coinvolgono solo gli attributi presenti in Ri. Ovvero, sono le dipendenze funzionali che continuano a valere quando consideriamo solo Ri** invece dell'intera R.

Legge 3: «non perderti dipendenze funzionali!»

Esempio

Consideriamo lo schema di relazione R(A, B, C, D) e un insieme di dipendenze funzionali del tipo

F = {
A → B
B → C
A → D
}

Consideriamo una decomposizione di R denominate D = R1 e R2 dove R1 **= {A, B, C} e R2={D}**

Come possiamo calcolare la proiezione πR1(F)?

Legge 3: «non perderti dipendenze funzionali!»

Algoritmo (1.b)

1. Calcoliamo la chiusura F+
2. Selezioniamo solo le dipendenze funzionali che riguardano gli attributi in R1

R1 = **{A, B, C}**

F = {
A → B
B → C
A → D
}

F+ = {
A → B ← dato
B → C ← dato
A → D ← dato
A->C ← IR3
A→BC ← IR2
A→BD ←IR2
A→BCD ← IR2
}

πR1(F) = {
A → B ← dato
B → C ← dato
A → D ← dato
A->C
A→BC ← ridondante
A→BD
A→BCD
}

πR1(F) = {
A → B
B → C
A->C
}

Legge 3: «non perderti dipendenze funzionali!»

Quando diciamo che una decomposizione è «Dependency preserving»? (Ovvero, come facciamo a dire che una decomposizione D rispetta la Legge 3 della ricetta?)

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

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

Legge 3: «non perderti dipendenze funzionali!»

Teorema di BD2 (non dimostrato):

«È sempre possibile trovare una decomposizione dependency-preserving D rispetto a F tale che ogni R_i in D è in BCNF oppure 3NF».

Di conseguenza, l'esercizio della prova scritta sarete costretti **a svolgerlo sempre!**

Algoritmo (1) di decomposizione dependency-preserving in schemi di relazione BCNF o 3NF.

Input:

R, uno schema di relazione R(A1, A2, ... An)
F, un insieme di dipendenze funzionali {→}

Ouput:

Una decomposizione D = {R1... Rn} e un insieme di proiezioni π_{Ri} tali che:

D soddisfa la 3NF

$$((\pi_{R1}(F)) \cup \dots \cup (\pi_{Rm}(F)))^{+} = F^{+}$$

Algoritmo (1) di decomposizione dependency-preserving in schemi di relazione BCNF o 3NF.

1. **Trovare una copertura minimale G di F (ovvero, esegui l'algoritmo 1.a).**

2. **Per ogni parte sinistra X di una dipendenza funzionale che appare nella copertura minimale G**, creare uno schema di relazione $\{X \cup A_1 \cup A_2 \cup \ldots \cup A_m\}$ in D dove $X \rightarrow A_1, X \rightarrow A_2, \ldots, X \rightarrow A_m$ sono le sole dipendenze in G aventi X come parte sinistra.

3. Mettere **in uno schema di relazione singolo tutti gli attributi rimanenti**, per garantire la proprietà di attribute-preservation.

Algoritmo (1) di decomposizione dependency-preserving in schemi di relazione BCNF o 3NF.

Alcune informazioni importanti:

1. Questo algoritmo non garantisce esplicitamente la preservazione delle chiavi del sistema relazionale;

2. Quando suddividiamo lo schema in più relazioni basate sulle dipendenze funzionali minimali, **le chiavi potrebbero essere frammentate tra più schemi (rompersi, come abbiamo visto la scorsa volta)**.

3. L'ultima fase dell'algoritmo **è un tentativo 'disperato' di preservare tutti gli attributi**, ma non assicura che una chiave dello schema originale sia ancora chiave in almeno uno dei nuovi schemi.

Algoritmo (1) di decomposizione dependency-preserving in schemi di relazione 3NF.

Ricerca di una copertura minimale?

Definizione: Una copertura minimale di un insieme F di dipendenze funzionali è:

«**un insieme minimale G** di dipendenze che è equivalente ad F (**G copre F e F Copre E)**»

Algoritmo (1.a) ricerca di una copertura minimale.

Un insieme di dipendenze funzionali F è minimale se:

1. **Ogni dipendenza in F ha un singolo attributo sul lato destro (atomizzazione);**

2. Non è possibile rimpiazzare una dipendenza X → A con una dipendenza Y → A dove Y è un sottoinsieme proprio di X ed avere ancora un insieme di dipendenze che è equivalente ad F.

3. **Non possiamo rimuovere una dipendenza da F ed ottenere un insieme di dipendenze equivalente ad F.**

Algoritmo (1.a) ricerca di una copertura minimale.

L’algoritmo 1.a in a nutshell

1. **Creare una copia di F nominadola G;**
2. **Atomizzare tutte le dipendenze funzionali in G;**
3. **Eliminare gli attributi ridondanti dalle dipendenze in G;**
4. **Eliminare le «dipendenze ridondanti» in G.**

Algoritmo (1.a) ricerca di una copertura minimale.

Bootstrap dell'algoritmo:

1. *porre* G:= F;

2. rimpiazzare ogni dipendenza funzionale $X \rightarrow \{A_1, A_2, \ldots, A_n\}$ in G, con **n** dipendenze funzionali $X \rightarrow A_1$, $X \rightarrow A_2$, ... , $X \rightarrow A_n$; ***(rendere le dipendenze atomiche)***

Algoritmo (1.a) ricerca di una copertura minimale.

Esempio di atomizzazione

Supponiamo di avere R(A, B, C, D, E) e l'insieme di dipendenze funzionali F = {A→BC,C→DE }

La dipendenza A → BC ha due attributi a destra (BC), quindi in G inseriamo due dipendenze:

A → B

A → C

Stessa cosa per C → DE, inseriamo in G:

C → D

C → E

Algoritmo (1.a) ricerca di una copertura minimale.

Passi iterativi (eliminazione degli attributi ridondanti)

3. *per ogni* dipendenza funzionale X → A in G
   per ogni attributo B che è un elemento di X
   {
   se { {G - {X → A} } ∪ { (X – {B}) → A} } è equivalente a G
   allora sostituire X → A con
   (X – {B}) → A in G;
   }

Algoritmo (1.a) ricerca di una copertura minimale.

Esempio rimozione attributi ridondanti

Supponiamo di avere R(A, B, C, D, E) e l'insieme di dipendenze funzionali G = G={AB→C,C→D,AC→E} **che abbiamo già atomizzato in precedenza.**

Esaminiamo la dipendenza AB → C.

- Proviamo a rimuovere B dalla parte sinistra e verifichiamo se A → C è equivalente a AB → C.
- Verifichiamo se (G - {AB → C}) ∪ {A → C} è equivalente a G
- **Se A → C e il resto delle dipendenze in G permettono di derivare C come prima, allora B era ridondante.**

Algoritmo (1.a) ricerca di una copertura minimale.

Esempio rimozione dipendenze ridondanti

Passi iterativi

4. *per ogni* dipendenza funzionale rimanente X → A in G
   {
   se { G – {X → A} } è equivalente a G
   allora rimuovere X → A da G;
   }

Algoritmo (1.a) ricerca di una copertura minimale.

Esempio rimozione dipendenze ridondanti

Supponiamo di avere R(A, B, C, D, E) e l'insieme di dipendenze funzionali G = G={A→C,C→D,A→D,AC→E} **che abbiamo già atomizzato in precedenza.**

Proviamo a rimuovere A→D e verifichiamo se **G–{A→D}** è ancora equivalente a **G. Dunque, se possiamo comunque ottenere D da A usando altre dipendenze, allora A→D è ridondante.**

Osserviamo che: A → C è presente in G. C → D è presente in G. Possiamo dedurre A → D con la transitività: Da A → C e C → D, otteniamo A → D. Abbiamo che A → D è ridondante e possiamo rimuoverlo.

Di conseguenza il nuovo insieme diventa G={A→C,C→D,A→D,AC→E}

Algoritmo (1) di decomposizione dependency-preserving in schemi di relazione BCNF o 3NF.

Ma… le chiavi che fine fanno?

Algoritmo (1) di decomposizione dependency-preserving in schemi di relazione BCNF o 3NF.

Recap

1. Questo algoritmo non garantisce esplicitamente la preservazione delle chiavi del sistema relazionale;

2. Quando suddividiamo lo schema in più relazioni basate sulle dipendenze funzionali minimali, **le chiavi potrebbero essere frammentate tra più schemi (rompersi, come abbiamo visto la scorsa volta)**.

3. L'ultima fase dell'algoritmo **è un tentativo 'disperato' di preservare tutti gli attributi**, ma non assicura che una chiave dello schema originale sia ancora chiave in almeno uno dei nuovi schemi.

Algoritmo (1) di decomposizione dependency-preserving in schemi di relazione BCNF o 3NF.

Dopo aver generato i nuovi schemi di relazione, dobbiamo ricalcolare le chiavi per assicurarci che almeno una di esse sia ancora valida.

Passi finali:

- **Per ogni schema Ri generato, troviamo la chiusura di ciascun possibile candidato a chiave.** Se un insieme di attributi in Ri chiude l'intero schema originale, allora è una chiave.

- **Se nessuno degli schemi include una chiave dell'originale, dobbiamo aggiungere una relazione contenente una chiave dello schema originale:**Questo evita la perdita di informazioni in termini di dipendenze e garantisce che possiamo ricostruire il database senza ambiguità.

Algoritmo (1) di decomposizione dependency-preserving in schemi di relazione BCNF o 3NF.

Trucco:

Dopo la decomposizione, controllare se almeno una delle nuove relazioni contiene una chiave candidata originale.

Se nessuna relazione contiene una chiave dell'originale, aggiungere una relazione supplementare contenente una chiave candidata originale.