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

Equivalenza di insieme di FD

Definizione:

Un insieme F di dipendenze funzionali **copre** un altro insieme E di dipendenze funzionali, se ogni DF in E è presente anche in F^+, cioè se ogni dipendenza in E può essere inferita a partire da F.

Definizione:

Due insiemi E ed F di dipendenze funzionali sono **equivalenti** se $E^+ = F^+$; ossia E è equivalente ad F se sussistono entrambe le condizioni: E **copre** F e F **copre** E.

Si può determinare **se F copre E** calcolando X^+ *rispetto* a F per ogni **DF X → Y in E**, e quindi verificando se questo X^+ comprende gli attributi presenti in Y.

Dimostrare l’equivalenza di 2 insiemi di FD

Per **verificare se i due insiemi di dipendenze funzionali F e G sono equivalenti**, dobbiamo dimostrare che entrambi determinano lo stesso insieme di dipendenze funzionali, ovvero che le chiusure di F e G coincidano

- Se F+ ⊆ G+ e G+ ⊆ F+, allora F ≡ G.

Passaggi metodologici:

- **Per ogni dipendenza funzionale X → Y presente in G, calcoliamo la chiusura di X+ rispetto ad F.**

Esempio

Consideriamo due insiemi di dipendenze funzionali F e G:

- $F = \{ CE \rightarrow D, E \rightarrow C, CF \rightarrow A, EF \rightarrow B \}$
- $G = \{ E \rightarrow ACD, CF \rightarrow AB, D \rightarrow F, BC \rightarrow E \}$

Verificare se F e G sono equivalenti. Applichiamo l'algoritmo seguente su F e G.

Esempio

Per ogni dipendenza funzionale X → Y presente in G, calcoliamo la chiusura di X+ rispetto ad F.

$F = \{ CE \rightarrow D, E \rightarrow C, CF \rightarrow A, EF \rightarrow B \}$

$G = \{ E \rightarrow ACD, CF \rightarrow AB, D \rightarrow F, BC \rightarrow E \}$

Esempio

Per ogni dipendenza funzionale X → Y presente in G, calcoliamo la chiusura di X+ rispetto ad F.

F = { CE → D, E → C, CF → A, EF → B }

G = { E → ACD, CF → AB, D → F, BC → E }

Non presenti in F

Calcoliamo {E}+, {CF}+, ~~{D}+ e {BC}+~~ rispetto a F. Ricaviamo:

- {E+} = { E per la IR1
    - C per la IR3
    - D per la IR3 ⇔ *E → C allora EE →D e dunque E → D*
    - }

Esempio

$F = \{ CE \rightarrow D,\ E \rightarrow C,\ CF \rightarrow A,\ EF \rightarrow B \}$

Calcoliamo {E}+, {CF}+, ~~{D}+ e {BC}+~~ rispetto a F. Ricaviamo:

{E}+ = {

E per la IR1

C per la IR3

D ⇔ Abbiamo *{C,E} nella chiusura di E → C allora {C,E}→D e dunque E → D*

}

{CF}+ = {

A

C,F per la IR1

}

Esempio

F = { CE → D, E → C, CF → A, EF → B }

Le chiusure delle FD di G rispetto a F sono:
{E}+ = {E,C,D}
{CF}+ = {C,F,A}

Le restanti chiusure sono.
{D}+ = {D}
{BC}+ = {BC}

Esempio

Ora operiamo al contrario. Per ogni dipendenza funzionale X → Y presente in F, calcoliamo la chiusura di X+ rispetto ad G.

F = { CE → D, E → C, CF → A, EF → B }

Non presenti in G

G = { E → ACD, CF → AB, D → F, BC → E }

Calcoliamo {E}+, {CF}+, ~~{CE}+ e {EF}+~~ rispetto a G. Ricaviamo:
{E+} = {
E ⇔ IR1
A,C,D ⇔ per FD diretta
F ⇔ IR3 (D→F)
B ⇔ poiché abbiamo C,F per FD diretta e per IR3
}

Esempio

Ora operiamo al contrario. Per ogni dipendenza funzionale X → Y presente in F, calcoliamo la chiusura di X+ rispetto ad G.

G = { E → ACD, CF → AB, D → F, BC → E }

Calcoliamo {E}+, {CF}+, ~~{CE}+ e {EF}+~~ rispetto a G. Ricaviamo:

{E+} = {E,A,C,D,F,B}

{CF}+ = {E,A,C,D,F,B}

Le restanti sono:

{CE}+ = {E,A,C,D,F,B}

{EF}+ = {E,A,C,D,B,F} ⇔ contiene E di conseguenza può implicare tutto quanto implica {E}

Esempio

Alla fine delle nostre inferenze ci troviamo nello stato seguente:

La chiusura di G rispetto a F è:

{E}+ = {C,E,A,D,F,B}
{CF}+ = {C,E,A,D,F,B}
{CE}+ = {C,E,A,D,F,B}
{EF}+ = {C,E,A,D,F,B}

$\subseteq$

La **chiusura di F rispetto a G** è:

{E+} = {E, C,D}
{CF}+ = {C,F,A}
{D}+ = {D}
{BC}+ = {B,C}

E' verificato che {F}+ ⊆ {G}+? SI, G copre F.
Il contrario è verificato? {G}+ ⊆ {F}+? No ci sono degli attributi in più!

Metodo rapido

Verifico se ogni X →Y in F è implicata dalle dipendenze funzionali in G (G copre F), **ossia se ogni X →Y è in G+.**

- **Rapidamente: Faccio la chiusura di G+ e controllo se contiene tutte le dipendenze funzionali di F;**
- **Poi faccio il contrario.**

F = { CE → D, E → C, CF → A, EF → B }

G = { E → ACD, CF → AB, D → F, BC → E }

G copre F?

- Per **CE → D** risulta $(CE)^{+G}$ = {CEADFB} quindi D ⊆ $(CE)^{+G}$
- Per **E → C** risulta $(E)^{+G}$ = {EACDFB} quindi C ⊆ $(E)^{+G}$
- Per **CF → A** risulta $(CF)^{+G}$ = {CFABED} quindi A ⊆ $(CF)^{+G}$
- Per **EF → B** risulta $(EF)^{+G}$ = {EFACDB} quindi B ⊆ $(EF)^{+G}$

Metodo rapido

Il contrario vale?

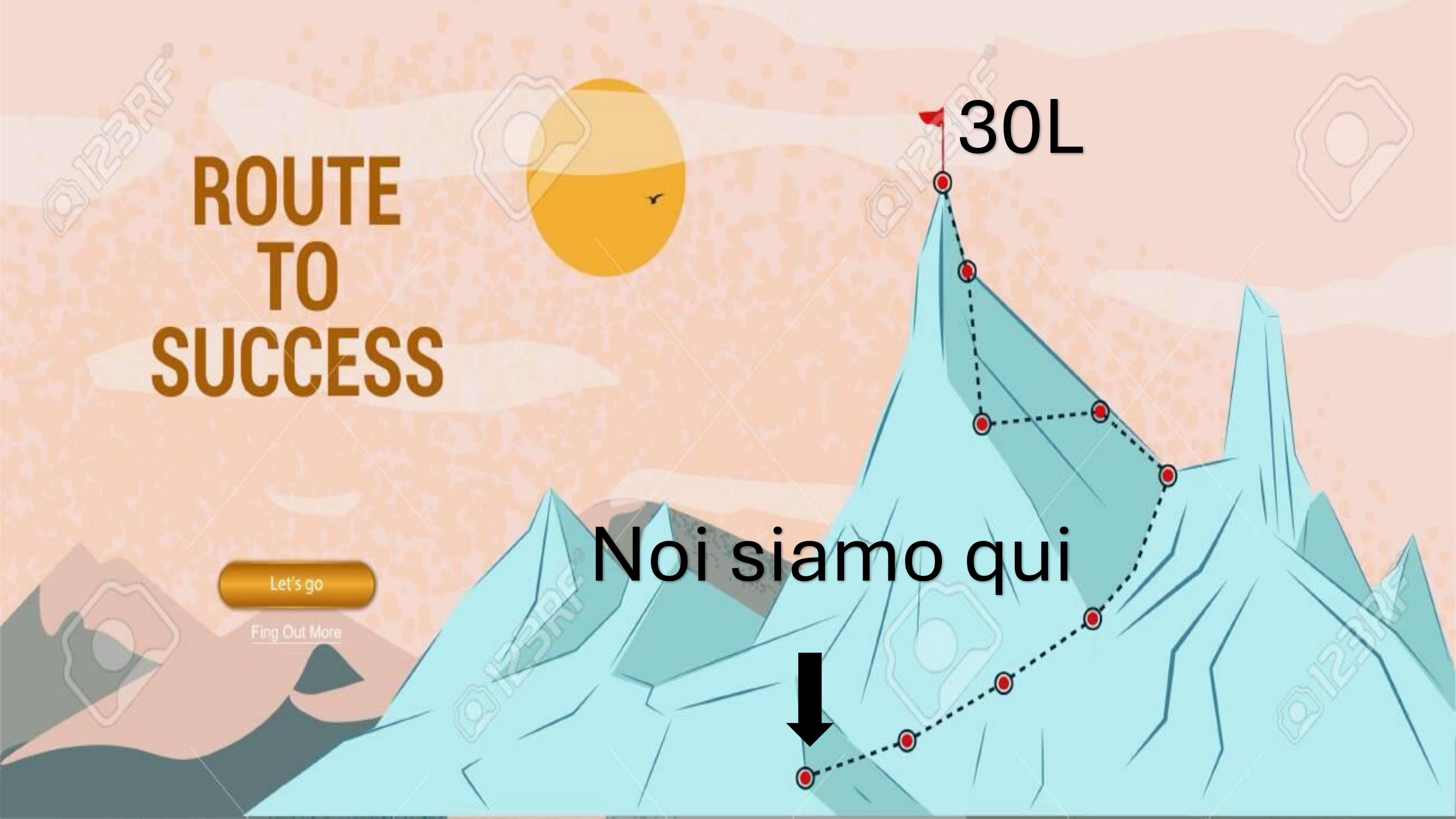
F = { CE → D, E → C, CF → A, EF → B }

G = { E → ACD, CF → AB, D → F, BC → E }

F copre G^

- Per **E → ACD** risulta $(E)^{+F} = \{ECD\}$ quindi $ACD \not\subseteq (E)^{+F}$
- Per **CF → AB** risulta $(CF)^{+F} = \{CFA\}$ quindi $AB \not\subseteq (CF)^{+F}$
- Per **D → F** risulta $(D)^{+F} = \{D\}$ quindi $F \not\subseteq (D)^{+F}$
- Per **BC → E** risulta $(BC)^{+G} = \{BC\}$ quindi $E \not\subseteq (BC)^{+F}$

Risulta che G copre F ma F non copre G di conseguenza i due insiemi non sono equivalenti.


ROUTE TO SUCCESS
Let's go
Fing Out More
30L
Noi siamo qui

Esercitazione

Verifichiamo se F e G sono equivalenti

$F = \{$
$A \rightarrow C,$
$AC \rightarrow D,$
$E \rightarrow AD,$
$E \rightarrow H.$
$\}$

$G = \{$
$A \rightarrow CD,$
$E \rightarrow AH.$
$\}$

METODO1:
Dimostro che le dipendenze funzionali in F sono derivabili da quelle in G (G copre F), e viceversa.

$A \rightarrow C \Leftrightarrow \{A\}^{+G} = \{A,C,D\} \Leftrightarrow C \subseteq \{A\}^{+G} \Leftrightarrow A \rightarrow CD \Rightarrow A \rightarrow C$

$E \rightarrow AD$ e $E \rightarrow H \Leftrightarrow E \rightarrow ADH \Rightarrow \{E\}^{+G} = \{E,A,H,C,D\} \Leftrightarrow \{ADH\} \subseteq \{E\}^{+G} \Leftrightarrow$
$E \rightarrow AH$ e $E \rightarrow D$

Riesco a mostrare che $AC \rightarrow D$ è derivabile dalle FD in G?
Per la IR2 $A \rightarrow CD \Leftrightarrow AC \rightarrow CCD \Rightarrow$ Per la decomposizione $\Rightarrow AC \rightarrow D$
E $AC \rightarrow C$

G di conseguenza copre F.

Esercitazione

Possiamo dire il contrario?

F = {
$A \rightarrow C$,
$AC \rightarrow D$,
$E \rightarrow AD$,
$E \rightarrow H$.
}

G = {
$A \rightarrow CD$,
$E \rightarrow AH$.
}

Dimostro che le dipendenze funzionali in G sono derivabili da quelle in F (F copre G).

Dobbiamo dimostrare che $\{A\}^{+F}$ contiene CD e AH tramite qualche regola di inferenza.

$\{A\}^{+F} = \{A,C\} \Rightarrow A \rightarrow C$ inoltre per la IR6 $AA \rightarrow D$ e dunque $A \rightarrow D \Rightarrow A \rightarrow CD$
$\{E\}^{+F} = \{E,A,D,H\} \rightarrow E \rightarrow AH$

Di conseguenza F copre G.

Esercitazione

Dati i due insiemi di dipendenze funzionali F e G, determinare se sono equivalenti.

F = {	G = {
B → C,	B → C,
B → A,	B → A,
AD → E,	AD → E,
BD → F	AD → F
}	

Verifichiamo se F copre G. Per ogni DF in G:

Esercitazione

F copre G?

F = {

$B \rightarrow C,$

$B \rightarrow A,$

$AD \rightarrow E,$

$BD \rightarrow F$

}

G = {

$B \rightarrow C,$

$B \rightarrow A,$

$AD \rightarrow E,$

$AD \rightarrow F$

Verifichiamo se F copre G. Per ogni DF in G $X \rightarrow Y$ ci assicuriamo che le DF di A riescano a rappresentare la parte destra Y.

Esercitazione

F copre G?

$F = \{$
$B \rightarrow C,$
$B \rightarrow A,$
$AD \rightarrow E,$
$BD \rightarrow F$
$\}$

$G = \{$
$B \rightarrow C,$
$B \rightarrow A,$
$AD \rightarrow E,$
$AD \rightarrow F$

$\{B\}^{+F} = \{B,C,A\}$ $\Leftrightarrow$ $B \rightarrow C$ e $B \rightarrow A$ in G sono coperte da F;
$\{AD\}^{+F} = \{A,D,E\}$ $\Leftrightarrow$ $AD \rightarrow E$ è in G è comperta da F;
Ora dobbiamo coprire $AD \rightarrow F$ in G. Poiché $B \rightarrow A$ sappiamo che $BD \rightarrow F \Rightarrow AD \rightarrow F$ per la IR3 (B è sempre determinato da A dunque possiamo sostituirlo)

Di conseguenza F copre G.

Esercitazione

ERRORE?

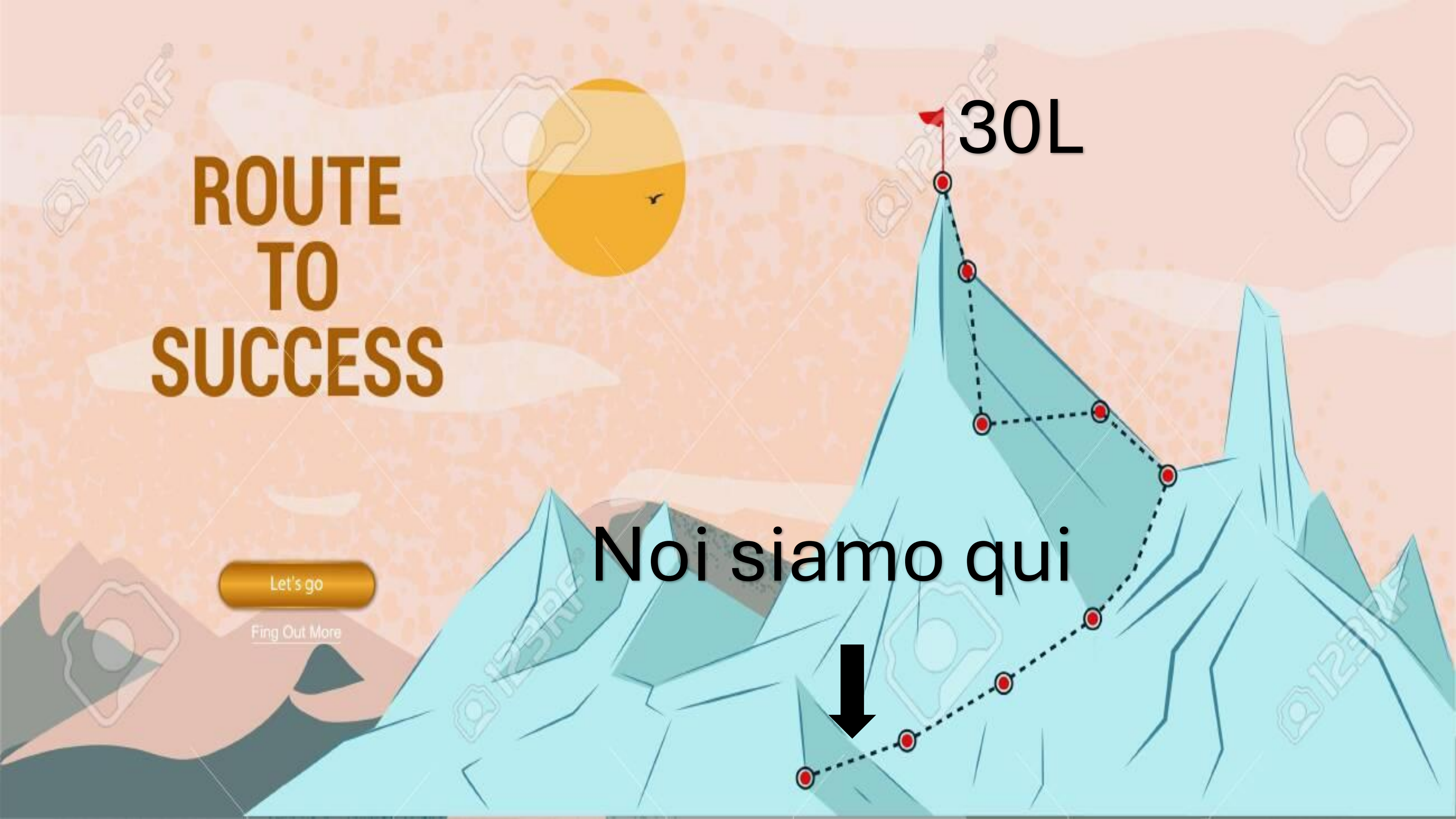
G copre F?

F = {
B → C,
B → A,
AD → E,
BD → F
}

G = {
B → C,
B → A,
AD → E,
AD → F

{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
30L
Noi siamo qui
Let's go
Fing Out More

Breve recap sui vostri obblighi (divini)

Misure di bontà informali

1. Fai in modo di avere una **semantica chiara degli attributi**;
2. **Riduci** il numero di valori **ridondanti** nelle tuple;
3. **Riduci** il numero di valori **null** nelle tuple;
4. **Non consentire tuple spurie**;

Linee guida estese!

5. Disegna uno schema di relazione facile da spiegare;
6. Disegna gli schemi di relazione in modo che non possano accadere insertion, deletion o modification anomalies.
7. Progetta gli schemi in modo da poter eseguire JOIN senza generare tuple spurie.

Bontà «Formali»

8. Definisci le dipendenze funzionali semanticamente «ovvie»
9. Calcola tutte le FD inferibili!
10. Normalizza lo schema (assicurandoti che sia dependencies preserving)!

La ricetta della buona decomposizione

Misure di bontà informali

Linee guida estese!

Bontà «Formali»

Cucina rispettando la ricetta!

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 $R = \{A_1, A_2, \ldots, A_n\}$ **che include tutti gli attributi** del database e data una decomposizione $D = \{R1, \ldots Rn\}$, 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)!»

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.

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 igliore 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 $\pi_{R_i}(F)$** (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** .

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 consideri solo Ri** invece dell'intera R.

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 l'unione delle proiezioni di F su ciascun R_i in D è equivalente a F, cioè:

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

Una cosa fantastica! <u>È sempre possibile trovare una decomposizione</u> dependency-preserving D rispetto a F tale che ogni R_i in D è in 3NF. ➔ Quindi l'esercizio della prova scritta sarete costretti a svolgerlo sempre!

Legge 3: «non perderti dipendenze funzionali!»

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

Input:

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

Ouput:

Una decomposizione D = {R1... Rn} tale che

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

Legge 3: «non perderti dipendenze funzionali!»

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

1. Trovare una copertura minimale G di F.

2. Per ogni parte sinistra X di una dipendenza funzionale che appare in G, creare uno schema di relazione $\{X \cup A_1 \cup A_2 \cup \dots \cup A_m\}$ in D dove $X \rightarrow A_1, X \rightarrow A_2, \dots, X \rightarrow A_m$ 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.

Legge 3: «non perderti dipendenze funzionali!»

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

Come trovo una copertura minimale?

Definizione: Una copertura minimale di un insieme E di dipendenze funzionali è **un insieme minimale** F di dipendenze che è equivalente ad E (E copre F e F Copre E).

Legge 3: «non perderti dipendenze funzionali!»

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

Come trovo una copertura minimale?

Un insieme di dipendenze funzionali F è minimale se:

1. **Ogni dipendenza in F ha un singolo attributo sul lato destro.**
2. Non è possibile rimpiazzare una dipendenza $X \rightarrow A$ con una dipendenza $Y \rightarrow A$ 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.**

Legge 3: «non perderti dipendenze funzionali!»

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

Bootstrap dell'algoritmo:

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

Legge 3: «non perderti dipendenze funzionali!»

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

Passi iterativi

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;
   }

Legge 3: «non perderti dipendenze funzionali!»

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

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;
   }