Basi di Dati 1

Simone Lidonnici

26 aprile 2024

Indice

1

Modello relazionale

Prodotto cartesiano tra domini

Il **dominio** è un insieme, anche infinito, di valori utilizzabili.
Presi $D_1, D_2, ..., D_k$ domini, non necessariamente diversi, il **prodotto cartesiano**, scritto:

$$D_1 \times D_2 \times ... \times D_k$$

é l'insieme:

$$\{(v_1, v_2, ..., v_k) | v_i \in D_i \ \forall i\}$$

Esempio:
$D_1 = \{a, b\}$
$D_2 = \{x, y, z\}$
$D_1 \times D_2 = \{(a, x), (a, y), (a, z), (b, x), (b, y), (b, z)\}$

Relazione

Una **relazione** r è un qualsiasi sottoinsieme di un prodotto cartesiano:

$$r \subseteq D_1 \times D_2 \times ... \times D_k$$

Il **grado** della relazione è pari al valore di k, cioè al numero dei domini.
In una relazione, ogni elemento $t \in r$ è una **tupla** che contiene numero di elementi pari al grado della relazione.
L'elemento i-esimo di una tupla viene indicato come $t[i]$ oppure $t.i$ con $i \in [1, k]$.
Una relazione ha una **cardinalità** pari al numero di tuple che la compongono.

Esempio:
$D_1 = \{\text{white,black}\}$
$D_2 = \{0, 1, 2\}$
$D_1 \times D_2 = \{(\text{white}, 0), (\text{white}, 1), (\text{white}, 2), (\text{black}, 0), (\text{black}, 1), (\text{black}, 2)\}$
La relazione $r = \{(\text{white}, 0), (\text{black}, 1), (\text{black}, 2)\}$ è una relazione di grado 2 e cardinalità 3
La tupla $t = (\text{white}, 0) \implies t[1] = \text{white}$ e $t[2] = 0$

1.1 Relazioni come tabelle

Le relazioni possono essere rappresentate come tabelle, in cui viene dato un nome ad ogni colonna.

Attributi e schema relazionale

Un **attributo** A è una coppia $(A, dom(A))$ in cui A è il nome dell'attributo e $dom(A)$ il suo dominio. Due attributi possono avere stesso dominio ma non stesso nome.
Uno **schema relazionale** è l'insieme di tutti gli attributi che definiscono la relazione associata allo schema stesso:

$$R(A_1, A_2, ..., A_k)$$

In uno schema relazionale, una tupla è una funzione che associa ad ogni attributo A un elemento del $dom(A)$. Con $t[A]$ indichiamo il valore della tupla corrispondente all'attributo A.
Un insieme di tuple r è un'**istanza di una relazione**.

Esempio:
$R = \{(\text{Nome, String}), (\text{Cognome, String}), (\text{Media, Real})\}$

Nome	Cognome	Media
Marco	Casu	28
Simone	Lidonnici	27

Schema di un database

Un insieme di relazioni distinte è uno **schema di un database**:

$$(R_1, R_2, ..., R_n)$$

Un database relazionale è uno schema di un database in cui sono definite le istanze $(r_1, r_2, ..., r_n)$ in cui r_i è un'istanza di R_i $\forall i$.

1.2 Scrivere tabelle corrette

I collegamenti tra i vari dati in schemi diversi vengono fatti attraverso i valori.
Esempio:

Matricola	Nome	Cognome
2041612	Marco	Casu
2061343	Simone	Lidonnici

Matricola	Voto	Esame
2041612	30	Sistemi
2061343	28	Algoritmi

In alcuni casi i potrebbero mancare delle informazioni, in questi casi si inserisce `NULL` come valore nella tupla, nel campo dell'attributo mancante. `NULL` è un valore che non appartiene a nessun dominio ma può rimpiazzare qualunque valore.
Esempio:

Matricola	Nome	Cognome
NULL	Nadia	Ge
2061343	NULL	Lidonnici

1.2.1 Vincoli di integrità

I dati devono avere dei vincoli de rispettare, questi vincoli possono essere:

- **Vincolo di dominio**: valori che devono essere per forza contenuti in un insieme specifico
- **Vincolo intra-relazionale**: sono vincoli tra valori di una stessa tupla
- **Vincolo inter-relazionale**: sono vincoli tra valori di diverse istanze
- **Vincolo di chiave primaria**: il valore deve essere diverso da `NULL` e unico negli attributi appartenenti alla chiave
- **Vincolo di esistenza**: il valore deve per forza essere diversa da `NULL`

Esempio:

Studenti:

Matricola	Nome	Cognome
2041612	Marco	Casu
2061343	Simone	Lidonnici

Voti:

Matricola	Esame	Voto	Lode
2061343	Probabilità	30	Si
2041612	Sistemi	28	NULL

In questo caso i vincoli sono:

- Vincolo di dominio: $18 \leq \text{Voto} \leq 30$
- Vincolo intra-relazionale: $\neg(\text{Voto} = 30) \implies \neg(\text{Lode} = \text{Si})$
- Vincolo inter-relazione: La matricola nello schema Voti deve esistere nello schema Studenti
- Vincolo di chiave primaria: La matricola nello schema Studenti deve per forza esistere ed essere unica
- Vincolo di esistenza: Il nome nello schema Studenti deve per forza esistere

1.2.2 Chiavi primarie

Chiave di una relazione

Una **chiave** di una relazione è un attributo o un gruppo di attributi che devono esistere ed appartenere ad una sola tupla.
Un insieme X di attributi dello schema R è una chiave di R se:

1. Per ogni istanza di R non esistono due tuple con gli stessi valori in tutti gli attributi di X, cioè $\forall t_1, t_2 \in r \;\; t_1[X] = t_2[X] \implies t_1 = t_2$
2. Per ogni sottoinsieme $X' \subset X$ possono esserci delle tuple che hanno tutti gli attributi di X' uguali ma sono diverse, cioè
$\forall X' \subset X \;\; \exists t_1, t_2 | t_1[X'] = t_2[X']$ ma $t_1 \neq t_2$

Esempio:

Lezione	Edificio	Aula	Orario
Basi di Dati	Chimica	2	16:00
Algebra	Informatica	2	17:00
Probabilità	Matematica	4	15:30

X = {Edificio, Aula, Orario} è una chiave primaria perché due lezioni non possono essere nello stesso edificio, aula e ora.
Togliendo però qualsiasi dei tre si possono avere delle lezioni diverse che abbiano gli altri attributi uguali (lezioni nello stesso edificio e aula ma ore diverse, nella stessa aula e ora ma edificio diverso o nello stesso edificio e ora ma aule diverse).

1.3 Dipendenze funzionali

Dipendenza funzionale

Una **dipendenza funzionale** è un insieme di attributi:

$$X, Y \subseteq R | X \to Y$$

Si dice "X determina Y" in cui X è il determinante e Y è il dipendente.

Esempio:
Flight(Code,Day,Pilot,Time)
Questo schema ha dei vincoli dati dal buonsenso:

- un volo con un codice parte sempre alla stessa
- un volo parte con un solo pilota, in un determinato orario e tempo

Questi vincoli determinano delle dipendenze funzionali:

- Code $\to$ Time
- {Day, Pilot, Time} $\to$ Code
- {Code, Day} $\to$ Pilot

Soddisfazione di una dipendenza funzionale

Un'istanza r di uno schema R soddisfa la dipendenza funzionale $X \to Y$ se:

1. La dipendenza è applicabile su R, cioè $X \subset R \land Y \subset R$
2. Due tuple in R che hanno gli stessi attributi X avranno anche gli stessi attributi Y, cioè $\forall t_1, t_2 \in r \;\; t_1[X] = t_2[X] \implies t_1[Y] = t_2[Y]$

2

Algebra relazionale

L'algebra relazionale è una notazione per specifiche domande (dette query) sul contenuto delle relazioni, le operazioni in algebra relazionale hanno una controparte in SQL. Per processare una query il DBMS trasforma SQL in una notazione simile all'algebra relazionale.
Il linguaggio per interrogare un database è un insieme di operatori binari o unitari applicati a una o più istanze per generare una nuova istanza.

2.1 Basi dell'algebra relazionale

L'algebra relazionale essendo un'algebra è composta da un dominio e delle operazioni che danno come risultato un elemento interno al dominio.
In questo caso i domini sono le relazioni, rappresentate come un insieme di tuple e i risultati delle operazioni sono un altro insieme di tuple.
Le query sono domande sull'istanza delle relazioni.
Ci sono 3 tipi di operazioni:

- modificare una relazione: proiezione, selezione e rinomina
- operazioni su insiemi: unione, intersezione e differenza
- combinare tuple di due relazioni: prodotto cartesiano, join e theta-join

2.2 Modifica di una relazione

2.2.1 Proiezione

Proiezione (π)

La **proiezione** esegue un taglio verticale su una relazione e sceglie un sottoinsieme degli attributi, creando una tabella con solo questi attributi.

$$\underset{A_1,\ldots,A_k}{\pi}(R)$$

Prende solo le colonne di un'istanza con attributi $A_1, ..., A_k$.

Esempio:
Customer:

Nome	C#	Città
Rossi	1	Roma
Rossi	2	Milano
Bianchi	3	Roma
Verdi	4	Roma
Rossi	5	Roma

Eseguendo $\underset{\text{Nome, Città}}{\pi}$(Customer):

Nome	Città
Rossi	Roma
Rossi	Milano
Bianchi	Roma
Verdi	Roma

2.2.2 Selezione

Selezione (σ)

La **selezione** fa un taglio orizzontale all'istanza di una relazione e sceglie tutte le tuple che rispettano una determinata condizione.

$$\underset{C}{\sigma}(R)$$

La condizione C è un'espressione booleana nella forma:

$$A\ \theta\ B \text{ o } A\ \theta\ a$$

In cui:

- θ è un operatore di comparazione, cioè $\{<, >, =, \leq, \geq, \text{ecc...}\}$
- A e B hanno lo stesso dominio
- a è un elemento del dominio di A usato come costante

Esempio:
Customer:

Nome	C#	Città
Rossi	1	Roma
Rossi	2	Milano
Bianchi	3	Roma
Verdi	4	Roma
Rossi	5	Roma

Eseguendo $\underset{\text{Town=Roma}}{\sigma}$(Customer):

Nome	C#	Città
Rossi	1	Roma
Bianchi	3	Roma
Verdi	4	Roma
Rossi	5	Roma

2.2.3 Rinomina

Rinomina (ρ)

La **rinomina** cambia il nome di un attributo in un altro.

$$\underset{A_1 \leftarrow A_2}{\rho}(R)$$

L'attributo A_1 viene rinominato in A_2.

2.3 Operazioni insiemistiche

Le operazioni insiemistiche possono essere fatte solo su istanze compatibili.
Due istanze sono compatibili se:

- hanno lo stesso numero di attributi
- gli attributi corrispondenti hanno lo stesso dominio

2.3.1 Unione

Unione ($\cup$)

L'**unione** crea una nuova istanza contenente tutte le tuple appartenenti a una delle due istanze unite.

$$r_1 \cup r_2$$

Bisogna stare attenti a non unire istanze con attributi che hanno lo stesso dominio ma che non c'entrano niente (ed esempio età e matricola).
Nel caso i cui gli attributi abbiano nomi diversi vengono presi i nomi della prima relazione.

Esempio:

Teachers:

Nome	Codice	Dipartimento
Rossi	1	Matematica
Bianchi	2	Fisica
Verdi	3	Inglese

Admin:

Nome	Codice	Dipartimento
Esposito	1	Matematica
Perelli	2	Informatica
Verdi	3	Inglese

Eseguendo AllStaff = Teachers $\cup$ Admin:

Nome	Codice	Dipartimento
Rossi	1	Matematica
Bianchi	2	Fisica
Verdi	3	Inglese
Esposito	1	Matematica
Perelli	2	Informatica

In questo caso abbiamo cancellato una riga che contiene (Verdi, 3, Inglese), per sistemarlo potremmo cambiare il codice degli insegnanti in T1,T2, ecc... e quello degli admin in A1,A2, ecc...

2.3.2 Differenza

Differenza (−)

La **differenza** crea una nuova istanza contenente tutte le tuple della prima che non sono presenti nella seconda. Si identifica con −.

$$r_1 - r_2$$

La differenza al contrario di unione e intersezione non è commutativa.

Esempio:

Students:

Nome	Codice	Dipartimento
Rossi	1	Matematica
Bianchi	2	Inglese
Verdi	3	Informatica

Admin:

Nome	Codice	Dipartimento
Esposito	4	Informatica
Bianchi	2	Inglese
Perelli	5	Matematica

Eseguendo Students − Admins:

Nome	Codice	Dipartimento
Rossi	1	Matematica
Verdi	3	Informatica

Eseguendo Admin − Students:

Nome	Codice	Dipartimento
Esposito	4	Informatica
Perelli	5	Matematica

2.3.3 Intersezione

Intersezione (∩)

L'**intersezione** crea una nuova istanza con le tuple in comune alle due istanze.

$$r_1 \cap r_2$$

Può anche essere definita come $r_1 - (r_1 - r_2)$.

2.4 Operazioni di combinazione

2.4.1 Prodotto cartesiano

Prodotto cartesiano ($\times$)

Il **prodotto cartesiano** crea una relazione con tutte le possibili combinazioni delle tuple della prima e della seconda relazione.

$$R_1 \times R_2$$

Esempio:

Customers:

Nome	C#	Città
Rossi	1	Roma
Rossi	2	Milano
Bianchi	3	Roma

Admin:

O#	CC#	A#	Pezzi
1	1	2	100
2	2	2	250
3	3	3	50

Eseguendo Orders = Customer $\times$ Order:

Nome	C#	Città	O#	CC#	A#	Pezzi
Rossi	1	Roma	1	1	2	100
Rossi	1	Roma	2	2	2	250
Rossi	1	Roma	3	3	3	50
Rossi	2	Milano	1	1	2	100
Rossi	2	Milano	2	2	2	250
Rossi	2	Milano	3	3	3	50
Binchi	3	Roma	1	1	1	100
Binchi	3	Roma	2	2	2	250
Binchi	3	Roma	3	3	3	50

In questo caso ho collegato dei Customer ad alcuni Order con C# diversi sbagliando e avendo alcun tuple in eccesso, quindi eseguo una selezione solo delle tuple che hanno C# uguale a CC# e poi successivamente una proiezione per eliminare l'attributo CC# ora superfluo.
Eseguendo Final = $\underset{\neg CC\#}{\pi}(\underset{C\#=CC\#}{\sigma}(\text{Customer} \times \text{Order}))$:

Nome	C#	Città	O#	A#	Pezzi
Rossi	1	Roma	1	2	100
Rossi	2	Milano	2	2	250
Binchi	3	Roma	3	3	50

2.4.2 Concatenazione

Concatenazione ($\bowtie$)

La **concatenazione** seleziona le tuple di un prodotto cartesiano che hanno gli stessi valori negli attributi in comune:

$$r_1 \bowtie r_2 = \underset{XY}{\pi}(\underset{C}{\sigma}(r_1 \times r_2))$$

Dove:

- $C = (R_1.A_1 = R_2.A_1) \wedge (R_1.A_2 = R_2.A_2) \wedge ... \wedge (R_1.A_k = R_2.A_k)$
- X è l'insieme degli attributi di r_1
- Y è l'insieme degli attributi di r_2 non in r_1
- Gli attributi doppi vengono automaticamente tolti

Ci sono alcuni casi speciali:

- Se non ci sono istanze che soddisfano le condizioni il risultato sarà l'insieme vuoto (che è comunque una relazione)
- Se le relazioni non hanno attributi in comune il risultato è semplicemente il prodotto cartesiano

Dobbiamo inoltre stare attenti che attributi con lo stesso nome abbiano lo stesso significato.
Esempio:

Customers:

Nome	C#	Città
Rossi	1	Roma
Rossi	2	Milano
Bianchi	3	Roma

Admin:

O#	CC#	A#	Pezzi
1	1	2	100
2	2	2	250
3	3	3	50
1	1	3	200

Eseguendo Orders = Customer $\bowtie$ Order:

Nome	C#	Città	O#	CC#	A#	Pezzi
Rossi	1	Roma	1	1	2	100
Rossi	1	Roma	1	1	3	200
Rossi	2	Milano	2	2	2	250
Binchi	3	Roma	3	3	3	50

2.4.3 Theta join

Theta join

Il **theta join** è una variazione del join in cui la condizione di selezione non è per forza il fatto che attributi con nome uguale abbiano valore uguale ma possiamo scegliere noi quali attributi collegare tra loro.

$$r_1 \underset{A\theta B}{\bowtie} r_2 = \underset{XY}{\pi}(\underset{A\theta B}{\sigma}(r_1 \times r_2))$$

In questo caso il join non collegherà due attributi con lo stesso nome ma collegherà l'attributo A e B in base alla condizione.

Esempio:

Customers:

Nome	C#	Città
Rossi	1	Roma
Rossi	2	Milano
Bianchi	3	Roma

Admin:

O#	CC#	A#	Pezzi
1	1	2	100
2	2	2	250
3	3	3	50

Eseguendo Orders = Customer $\underset{\text{C\#=CC\#}}{\bowtie}$ Order:

Nome	C#	Città	O#	CC#	A#	Pezzi
Rossi	1	Roma	1	1	2	100
Rossi	1	Roma	1	1	3	200
Rossi	2	Milano	2	2	2	250
Binchi	3	Roma	3	3	3	50

2.5 Quantificazioni universali e esistenziali

Quando si scrive una query possiamo interscambiare la quantificazione universale con quella esistenziale, negando la proposizione:

$$\forall x \to \varphi(x) = \nexists x \to \neg\varphi(x)$$

Bisogna ricordare che il contrario di "sempre" è "esiste almeno un caso in cui è falso" e non "è falso in ogni caso".

$$\neg(\forall x \to \varphi(x)) \neq \nexists x \to \varphi(x)$$
$$\neg(\forall x \to \varphi(x)) = \exists x \to \neg\varphi(x)$$

Esempio:

Customers:

Nome	C#	Città
Rossi	1	Roma
Rossi	2	Milano
Bianchi	3	Roma

Admin:

C#	A#	Pezzi
1	1	50
1	1	200
2	2	150
3	3	125

Se vogliamo trovare i nomi e città delle persone che hanno sempre fatto ordini da più di 100 pezzi dobbiamo eliminare dal totale le persone che hanno fatto anche solo una volta un ordine da meno di 100 pezzi:
$\underset{\text{Nome,Città}}{\pi}((\text{Customer} \bowtie \text{Order}) - \underset{\text{n° pezzi}\leq 100}{\sigma}(\text{Customer} \bowtie \text{Order}))$

Employees:

Nome	C#	Sezione	Salario	Sup#
Rossi	1	B	100	3
Pirlo	2	A	200	3
Bianchi	2	A	500	NULL
Neri	4	B	150	1

Per trovare i dipendenti con stipendio maggiore dei supervisori bisogna fare prima un join della tabella con una copia di se stessa collegando il codice dei supervisori con i codici dei dipendenti. Per non confonderci rinominiamo tutti i valori della copia con una C davanti.
CEmployees = Employees
Employees2 = Employees $\underset{\text{Employees.Supervisor\#=CEmployees.C\#}}{\bowtie}$ CEmployees:

Nome	C#	Sez	Sal	Sup#	CNome	C#	CSez	CSal	CSup#
Rossi	1	B	100	3	Bianchi	3	A	500	NULL
Pirlo	2	A	200	3	Bianchi	3	A	500	NULL
Neri	4	B	150	1	Rossi	1	B	100	3

Successivamente bisogna selezionare solo le tuple in cui lo stipendio dei supervisori è minore dei dipendenti e poi proiettare solo nome, codice e salario del dipendente.
Final = $\underset{\text{Nome,C\#,Salario}}{\pi}(\underset{\text{Sal}>\text{CSal}}{\sigma}(\text{Employees2}))$

Nome	C#	Sal
Neri	4	150

3

Progettare un database relazionale

Per creare un database dobbiamo analizzare gli attributi che deve contenere il database e capire se è meglio fare uno schema singolo o multipli schemi collegati. È preferibile dividere gli attributi in più schemi per evitare problemi di inserimento (se alcuni dati possono essere aggiunti successivamente), cancellazione (se bisogna cancellare una tupla ma lasciare l'esistenza di alcuni attributi) o ridondanza. Per progettare un buon database bisogna separare i concetti in schemi diversi. Possiamo identificare i concetti rappresentati in una relazione dalla chiave.

3.1 Notazioni

- Uno schema relazionale viene definito con R ed è un insieme di attributi: $R = \{A_1, A_2, ..., A_n\}$ oppure $R = A_1, A_2, ..., A_n$
- Un attributo viene definito con le prime lettere dell'alfabeto: $A, B, \ldots$
- - Un insieme di attributi viene definito con le ultime lettere dell'alfabeto: $X, Y, \ldots$
- L'unione di due insiemi $X \cup Y$ viene definita con XY
- Una tupla t su R è una funzione che associa ad ogni attributo di $A_i \in R$ un valore $t[A_i]$ nel dominio di A_i

3.2 Dipendenza funzionali

Dipendenza funzionali

Una dipendenza funzionale è un insieme di attributi:

$$X, Y \subseteq R | X \to Y$$

Si dice "X determina Y" in cui X è il determinante e Y è il dipendente.
L'insieme delle dipendenze funzionali di uno schema si identifica con F.
Un'istanza r è legale rispetto ad F se:

$$\forall (X \to Y) \in F \implies r \text{ SAT } (X \to Y)$$

Se $F = \{A \to B, B \to C\}$ allora è come se in F ci fosse anche $A \to C$.

Esempio:

Customers:

A	B	C
1	1	1
1	0	1
1	1	1

Soddisfa le dipendenza perché:
$t_1[AB] = t_3[AB] \implies t_1[C] = t_3[C]$

Admin:

A	B	C
1	1	1
1	0	1
1	1	0

Non soddisfa le dipendenza perché:
$t_1[AB] = t_3[AB]$ ma $t_1[C] \neq t_3[C]$

3.3 Chiusura di un insieme di dipedenza funzionali

Definizione di F^+

Dato uno schema relazionale R e un insieme di dipendenze funzionali F, la chiusura di F è l'insieme di tutte le dipendenze funzionali soddisfatte da tutte le tuple in R. Viene denotata con F^+.

$$F^+ = \{X \to Y | \forall r \text{ legale} \implies r \text{ SAT } (X \to Y)\}$$

Da questo possiamo dire che:

$$F \subseteq F^+$$
$$Y \subseteq X \implies (X \to Y) \in F^+$$

3.3.1 Assiomi di Armstrong

Definizione di F^A

Definiamo un altro insieme di dipendenze funzionali F^A creato con degli assiomi partendo da F:

- Inclusione iniziale: $(X \to Y) \in F \implies (X \to Y) \in F^A$
- Riflessività: $Y \subseteq X \subseteq R \implies (X \to Y) \in F^A$
- Aumento: $(X \to Y) \in F^A \implies (XZ \to YZ) \in F^A \forall Z \in R$
- Transitività: $\left.\begin{array}{l}(X \to Y) \in F^A \\ (Y \to Z) \in F^A\end{array}\right\} \implies X \to Z \in F^A$

Da questi primi 4 assiomi si ottengono implicitamente altre 3 proprietà:

1. **Unione:**
$$\left.\begin{array}{l}(X \to Y) \in F^A \\ (X \to Z) \in F^A\end{array}\right\} \implies (X \to YZ) \in F^A$$
Dimostrazione:
$$\left.\begin{array}{l}(X \to Y) \in F^A \xrightarrow{\text{aumento}} (XX \to XY) \in F^A \\ (X \to Z) \in F^A \xrightarrow{\text{aumento}} (XY \to ZY) \in F^A\end{array}\right\} \xrightarrow{\text{trans}} (X \to YZ) \in F^A$$

2. **Decomposizione:**
$$\left.\begin{array}{l}(X \to Y) \in F^A \\ Z \subseteq Y\end{array}\right\} \implies (X \to Z) \in F^A$$
Dimostrazione:
$$\left.\begin{array}{l}Z \subseteq Y \xrightarrow{\text{rifl}} (Y \to Z) \in F^A \\ (X \to Y) \in F^A\end{array}\right\} \xrightarrow{\text{trans}} (X \to Z) \in F^A$$

3. **Pseudotrasitività:**
$$\left.\begin{array}{l}(X \to Y) \in F^A \\ (WY \to Z) \in F^A\end{array}\right\} \implies (WX \to Z) \in F^A$$
Dimostrazione:
$$\left.\begin{array}{l}(X \to Y) \in F^A \xrightarrow{\text{aumento}} (WX \to WY) \in F^A \\ (WY \to Z) \in F^A\end{array}\right\} \xrightarrow{\text{trans}} (WX \to Z) \in F^A$$

3.4 Chiusura di un insieme di attributi

Definizione di X^+

Dato uno schema relazionale R con F insieme delle dipendenze su R e $X \subset R$ la chiusura di X rispetto ad F definita come X_F^+ o X^+:

$$X_F^+ = \{A | (X \to A) \in F^A\}$$

Ovviamente per riflessività:

$$X \subseteq X_F^+$$

Dato uno schema relazionale R con F insieme delle dipendenze funzionali su R:

$$(X \to Y) \in F^A \iff Y \subseteq X^+$$

Dimostrazione:
$Y = A_1, A_2, ..., A_n$

- $Y \subseteq X^+ \implies (X \to A_i) \in F^A \quad \forall A_i \in Y \xrightarrow{\text{unione}} (X \to Y) \in F^A$
- $(X \to Y) \in F^A \xrightarrow{\text{decomp}} (X \to A_i) \in F^A \quad \forall A_i \in Y \implies A_i \in X^+ \forall i \implies Y \subseteq X^+$

3.4.1 Calcolare la chiusura di X

Dato uno schema relazionale R con F insieme della dipendenze funzionali, X^+ si calcola tramite un algoritmo:

Algoritmo: Calcolo di X^+

Input:

- R: schema
- F: insieme di dipendenze
- X: insieme di attributi

```
def CalcolaChiusura(R, F, X):
    Z=X
    S = {A|∃(Y → V) ∈ F, Y ⊆ Z, A ∈ V}
    while S⊄Z :
        Z=Z∪S
        S = {A|∃(Y → V) ∈ F, Y ⊆ Z, A ∈ V}
    return Z
```

Esempio:
$R = ABCDEH$
$F = \{AB \to CD, EH \to D, D \to H\}$
Seguendo l'algoritmo calcoliamo la chiusura di A, D, AB:

- $A^+ = A$
- $D^+ = DH$
- $AB^+ = ABCDH$

3.4.2 Dimostrazione dell'algoritmo

Dato un insieme di attributi X, l'insieme Z uscito dall'algoritmo è uguale alla chiusura di X:

$$Z = X^+$$

Dimostrazione per doppia inclusione:
Definisco:

$$Z_i = \Big\{ Z \text{ dopo } i \text{ cicli while} \quad \implies Z_{i+1} = Z_i \cup S_i$$
$$Z_f = Z_{\text{finale}}$$

1. $Z_f \subseteq X^+$
   Dimostro per induzione che se $Z_i \subseteq X^+$ allora anche $Z_{i+1} \subseteq X^+$:

   1.1 Caso base: $i = 0$
   $Z_0 = X \subseteq X^+$

   1.2 Passo induttivo:
   $Z_i \subseteq X^+ \implies Z_{i+1} \subseteq X^+$

   1.3 Dimostrazione induttiva:
   $\forall A \in Z_{i+1}$ ci sono due casi possibili:

   - $A \in Z_i \xrightarrow{\text{passo induttivo}} A \in X^+$
   - $A \in S_i \implies \begin{cases} \exists (Y \to V) \in F \\ Y \subseteq Z_i \\ A \in V \end{cases} \implies Y \subseteq Z_i \subseteq X^+ \implies$
     $(X \to Y) \in F^A \xrightarrow{\text{trans}} (X \to V) \in F^A \implies V \subseteq X^+ \implies$
     $A \in X^+$

2. $X^+ \subseteq Z_f$

	Z_f	R\Z_f
t_1	11...1	01...0
t_2	11...1	10...1

uguali su Z_f

Devo controllare se l'istanza è legale per ogni dipendenza di F.
$\forall (V \to W) \in F$ ci sono due casi possibili:

- $V \cap R \backslash Z_f \neq 0 \implies t_1[V] \neq t_2[V]$
- $V \subseteq Z_f \implies W \subseteq S_f \subseteq Z_f \implies t_1[W] = t_2[W]$

Ora che ho dimostrato che l'istanza è legale:
$\forall A \in X^+ \implies (X \to A) \in F^A \implies X = Z_0 \subseteq Z_f \implies A \in Z_f$

3.5 $\mathbf{F^+ = F^A}$

Dato uno schema relazionale R con F insieme delle dipendenze funzionali:

$$F^+ = F^A$$

Dimostrazione per doppia inclusione:

1. $F^A \subseteq F^+$
   Presa una dipendenza funzionale $(X \to Y) \in F^A$ definisco:

$$F_i^A = \begin{cases} \text{dipendenze funzionali} \\ \text{ottenute con } i \\ \text{applicazioni degli assiomi} \\ \text{di Armstrong} \end{cases}$$

$$F_0^A \subseteq F_1^A \subseteq F_2^A \subseteq ... \subseteq F^A$$

   Per induzione devo dimostrare che se una dipendenza ottenuta con i applicazioni degli assiomi di Armstrong appartiene ad F^+ allora anche la dipendenza ottenuta con $i+1$ applicazioni degli assiomi di Armstrong appartiene ad F^+:

   1.1 Caso base: $i = 0$
   $F_0^A = F \subseteq F^+$

   1.2 Passo induttivo:
   $F_i^A \subseteq F^+ \implies F_{i+1}^A \subseteq F^+$

   1.3 Dimostrazione induttiva:
   L'ultimo passaggio (per passare da i a $i+1$) può essere qualsiasi dei 3 assiomi:

   - Riflessività:
     $Y \subseteq X \implies (X \to Y) \in F_{i+1}^A$
     $t_1[X] = t_2[X] \implies t_1[Y] = t_2[Y] \implies (X \to Y) \in F^+$
   - Aumento:
     $(V \to W) \in F_i^A \implies (VZ \to WZ) \in F_{i+1}^A$
     $t_1[VZ] = t_2[VZ] \implies \begin{cases} t_1[V] = t_2[V] \\ t_1[Z] = t_2[Z] \end{cases} \implies \begin{cases} t_1[W] = t_2[W] \\ t_1[Z] = t_2[Z] \end{cases} \implies$
     $t_1[WZ] = t_2[WZ] \implies$
     $(VZ \to WZ) \in F^+$
   - Transitività:
     $\left.\begin{array}{l}(X \to Y) \in F_i^A \\ (Y \to Z) \in F_i^A\end{array}\right\} \implies (X \to Z) \in F_{i+1}^A$
     $t_1[X] = t_2[X] \implies t_1[Y] = t_2[Y] \implies t_1[Z] = t_2[Z] \implies (X \to Z) \in F^+$

2. $F^+ \subseteq F^A$

	X^+	R\X^+
t_1	11...1	01...0
t_2	11...1	10...1

uguali su X^+

Devo controllare se l'istanza è legale per ogni dipendenza di F.
$\forall (V \to W) \in F$ ci sono due casi possibili:

- $V \cap R\backslash X^+ \neq 0 \implies t_1[V] \neq t_2[V]$
- $\left.\begin{array}{l} V \subseteq X^+ \implies (X \to V) \in F^A \\ (V \to W) \in F^A \end{array}\right\} \xRightarrow{\text{trans}} (X \to W) \in F^A \implies W \subseteq X^+$

Ora che ho dimostrato che l'istanza è legale:
$\forall (X \to Y) \in F^+ \implies Y \subseteq X^+ \implies (X \to Y) \in F^A$

3.6 Chiavi di uno schema

Definizione di chiave

Una chiave è un insieme di attributi $X \subseteq R$ in cui:

- $(X \to R) \in F^+$
- $\nexists X' \subseteq X | (X' \to R) \in F^+$

4

Terza forma normale

Definizione di schema in 3NF

Dato uno schema relazionale R con F insieme delle dipendenze funzionali su R, R è in **terza forma normale** (3NF) se:

$$\forall (X \to A) \in F^+, A \notin X$$

Una dipendenza è in 3NF se X contiene una chiave o Y è contenuto in una chiave.

$$X \supseteq K \vee Y \subseteq K$$

Per controllare se uno schema è scritto in 3NF devo controllare tutte le dipendenze ottenute decomponendo quelle in F in modo che il dipendente sia un attributo singolo, basta che una non vada bene e tutto lo schema non è scritto in 3NF.
Possiamo dire anche che uno schema è in 3FN se non contiene dipendenze **parziali** né dipendenze **transitive**:
$\begin{cases} F = \{AB \to C, B \to C\} \\ K = AB \end{cases} \implies B \to C$ è una dipendenza parziale perché dipende parzialmente da una chiave
$\begin{cases} F = \{A \to B, B \to C\} \\ K = A \end{cases} \implies B \to C$ è una dipendenza transitiva perché dipende in modo transitivo da una chiave

Esempi:
$R = ABCD$
$F = \{A \to B, B \to CD\}$
A è l'unica chiave perché determina B e per transitività anche CD
Devo controllare:

- $A \to B$: va bene perché A è chiave
- $B \to C$: non va bene perché B non è chiave e C non è contenuto in una chiave
- $B \to D$

$R = ABCD$
$F = \{AC \to B, B \to AD\}$
AC e BC sono chiavi
Devo controllare:

- $AC \to B$: va bene perché AC è chiave
- $B \to A$: va bene perché A è contenuto nella chiave AC
- $B \to D$: non va bene perché B non è chiave e D non è contenuto in una chiave

4.1 Forma normale di Boyce-Codd

Definizione di schema in BCNF

Dato uno schema relazionale R con F insieme delle dipendenze funzionali, R è in Boyce-Codd NF (BCNF) se tutti i determinanti contengono una chiave.

$$(X \to Y) \in F^+ | K \subseteq X$$

Tutti gli schemi in Boyce-Codd NF sono anche in 3NF ma non è sempre vero il contrario. A differenza della 3NF non è sempre possibile decomporre uno schema non in BCNF in sottoschemi in BCNF preservando tutte le dipendenze.

4.2 Trasformare uno schema in 3NF

Uno schema non in 3NF può essere decomposto in un insieme di schemi in 3NF che devono rispettare delle regole:

- devono preservare le dipendenze funzionali applicate ad ogni istanza dello schema originale (preservare F)
- bisogna poter ricostruire tramite join naturale ogni istanza legale dello schema originale senza aggiungere informazioni (join senza perdita)

Esempio:
$F = \{A \to B, B \to C\}$ non in forma normale perché $B \to C$ è una dipendenza transitiva
Posso scomporlo in due modi:

- $\begin{cases} R_1 = AB \text{ con } F_1 = \{A \to B\} \\ R_2 = BC \text{ con } F_2 = \{B \to C\} \end{cases}$
- $\begin{cases} R_1 = AB \text{ con } F_1 = \{A \to B\} \\ R_2 = AC \text{ con } F_2 = \{A \to C\} \end{cases}$

Il secondo modo non va bene perché quando li mettiamo insieme la dipendenza $B \to C$ non è soddisfatta.

4.3 Preservare F

Definizione di decomposizione

Dato uno schema relazionale R, una decomposizione è un insieme di sottinsiemi di R tali che:

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

Questa non è una partizione di R perché i sottoschemi non sono disgiunti.
Presa un'istanza r l'istanza di un sottoschema r_i:

$$r_i = \underset{R_i}{\pi}(r)$$

Decomposizione che preserva F

Dato uno schema relazionale R, una decomposizione $\rho = R_1, ..., R_k$ preserva F se:

$$F \equiv G = \bigcup_{i=0}^{k} \underset{R_i}{\pi}(F)$$

$$\underset{R_i}{\pi}(F) = \{(X \to Y) \in F^+ | XY \subseteq R_i\}$$

Una singola dipendenza $(X \to Y) \in F$ è preservata se $Y \subseteq X_G^+$.

4.3.1 Teorema sulla chiusura

Teorema sulla chiusura

Dati due insiemi di dipendenze funzionali F, G:

$$F \subseteq G^+ \iff F^+ \subseteq G^+$$

Dimostrazione per doppia implicazione:

1. $F^+ \subseteq G^+ \implies F \subseteq G^+$
   $F \subseteq F^+ \subseteq G^+ \implies F \subseteq G^+$
2. $F \subseteq G^+ \implies F^+ \subseteq G^+$
   Scrivo $g \xrightarrow{A} f$ per dire che posso ottenere f applicando gli assiomi di Arstrong su g
   $\forall f \in F \implies \exists g | g \xrightarrow{A} f \implies G \xrightarrow{A} F \xrightarrow{A} F^+ \implies F^+ \subseteq G^+$

4.4 Calcolare X_G^+

Dato uno schema relazionale R con F insieme della dipendenze funzionali e G rispetto ad F, X^+ si calcola tramite un algoritmo (non conosciamo G):

Algoritmo: Calcolo di X_G^+

Input:

- R: schema
- F: insieme di dipendenze
- X: insieme di attributi
- ρ: decomposizione

```
def CalcolaChiusuraG(R,F,X,ρ):
    Z=X
    S={}
    for i in range(1,k) :
        S = S ∪ (Z ∩ R_i)_F^+ ∩ R_i
    while S⊈Z :
        Z=Z∪S
        for i in range(1,k) :
            S = S ∪ (Z ∩ R_i)_F^+ ∩ R_i
    return Z
```

Dimostrazione per doppia inclusione:

1. $Z_f \subseteq X_G^+$

   1.1 Caso base: $i = 0$
   $Z_0 = X \subseteq X_G^+$

   1.2 Passo induttivo:
   $Z_i \subseteq X_G^+ \implies Z_{i+1} \subseteq X_G^+$

   1.3 Dimostrazione induttiva:
   $\forall A \in Z_{i+1}$ ci sono due casi possibili:

   - $A \in Z_i \xrightarrow{\text{passo induttivo}} A \in X_G^+$
   - $A \in S_i \implies A \in [(Z_i \cap R_j)_F^+ \cap R_j] \implies \begin{cases} A \in R_j \\ A \in (Z_i \cap R_j)_F^+ \end{cases} \implies (Z_i \cap R_j \to A) \in F^A \implies (Z_i \cap R_j \to A) \in \underset{R_j}{\pi}(F) \subseteq G \implies \left.\begin{matrix} (X \to Z_i \cap R_j) \in G^A \\ (Z_i \cap R_j \to A) \in G^A \end{matrix}\right\} \xRightarrow{\text{trans}} (X \to A) \in G^A \implies A \in X_G^+$

2. $X_G^+ \subseteq Z_f$
   $X \subseteq Z_f \implies X_G^+ \subseteq (Z_f)_G^+$, se supponiamo di conoscere G possiamo applicare l'algoritmo per calcolare X_F^+ ma su G.
   Ogni elemento $A \in S$ è anche in Z perché:
   Se esiste $(Y \to V) \in G \implies YV \subseteq R_i \implies Y \subseteq (R_i \cap Z_f) \overset{\text{rifl}}{\implies}$
   $(R_i \cap Z_f \to Y) \in G^A \overset{\text{decomp}}{\implies} (R_i \cap Z_f \to A) \in G^A \implies$
   $A \in (R_i \cap Z_f)^+ \cap R_i \implies A \in Z_f$
   Quindi $Z_f = (Z_f)_G^+ \implies X_G^+ \subseteq Z_f$

4.5 Equivalenza tra insiemi di dipendenze funzionali

$F^+ = G^+$

Dati due insiemi di dipendenze funzionali F, G:

$$F \equiv G \iff F^+ = G^+$$

G e F non sono uguali.

$$\left.\begin{matrix} F^+ \subseteq G^+ \\ G^+ \subseteq F^+ \end{matrix}\right\} \iff \begin{cases} F \subseteq G^+ \\ G \subseteq F^+ \end{cases}$$

Dimostrazione tramite doppia inclusione:

1. $G \subseteq F^+$
   $G = \bigcup_{i=0}^{k} \pi_{R_i}(F) \implies G \subseteq F^+$

2. $F \subseteq G^+$
   per dimostrarlo utilizziamo un algoritmo:

Algoritmo: Calcolo se $F \subseteq G^+$

Input:

- R: schema
- F: insieme di dipendenze
- X: insieme di attributi
- ρ: decomposizione

```
def IsSottoinsieme(R,F,X,ρ):
    for (X → Y) in F :
        X_G^+=CalcolaChiusuraG(R,F,X,ρ)
        if Y⊄ X_G^+ :
            return False
        return True
```

4.6 Trovare le chiavi di uno schema

Per trovare la chiavi bisogna trovare un attributo o un insieme di attributi tali che la sua chiusura sia R e non ci siano sottoinsiemi tali che la loro chiusura sia R:

$$X = K \iff X^+ = R$$
$$\nexists X' \subseteq X | (X')^+ = R$$

Ci sono delle chiusure che si possono saltare:

- Se un attributo non è determinato da nulla appartiene a tutte le chiavi
- Se un attributo non determina niente, ma è solo determinato non appartiene a nessuna chiave

Per provare a calcolarle in modo facile seguiamo dei passaggi:

1. Calcoliamo $X_i = R - (W - V)$ per ogni dipendenza $V \to W$
2. Calcoliamo $X = (X_1 \cap ... \cap X_n)$
3. Calcoliamo X^+ e se il risultato è:
   - R allora X è l'unica chiave
   - diverso da R allora X non è chiave ed esiste più di una chiave

4.6.1 Teorema di unicità della chiave

Teorema di unicità della chiave

Dato uno schema R, possiamo controllare se una chiave è unica scrivendo:

$$X = \bigcap_{(V \to W) \in F} R - (W - V)$$

Calcoliamo X^+ e se:

- $X^+ = R \implies X$ unica chiave
- $X^+ \neq R \implies$ esiste più di una chiave e X non è superchiave

Esempio:
$R = ABCDE$
$F = \{AB \to C, AC \to B, D \to E\}$
$X = (ABCDE - (C - AB)) \cap (ABCDE - (B - AC)) \cap (ABCDE - (E - D)) = ABDE \cap ACDE \cap ABCD = AD$
$X^+ = AD^+ = ADE \neq R \implies$ esisto più chiavi

4.7 Join senza perdite

Decomposizione don join senza perdite

Dato uno schema relazionale R, una decomposizione $\rho = R_1, R_2, ..., R_k$ ha un join senza perdita se per ogni istanza r di R:

$$r = \underset{R_1}{\pi}(r) \bowtie \underset{R_2}{\pi}(r) \bowtie ... \bowtie \underset{R_k}{\pi}(r) = m_\rho(r)$$

Proprietà di una decomposizione

La decomposizione di R, $m_\rho(r)$ ha delle proprietà:

- $r \subseteq m_\rho(r)$
- $\underset{R_i}{\pi}(m_\rho(r)) = \underset{R_i}{\pi}(r)$
- $m_\rho(m_\rho(r)) = m_\rho(r)$

Dimostrazione:

1. $\forall t \in r \implies t \in (t[R_1] \bowtie ... \bowtie t[R_k]) \subseteq (\underset{R_1}{\pi}(r) \bowtie ... \bowtie \underset{R_k}{\pi}(r)) = m_\rho(r)$
2. $\underset{R_i}{\pi}(m_\rho(r)) = \underset{R_i}{\pi}(r)$
   - $\underset{R_i}{\pi}(r) \subseteq \underset{R_i}{\pi}(m_\rho(r)) \implies$ punto 1
   - $\underset{R_i}{\pi}(m_\rho(r)) \subseteq \underset{R_i}{\pi}(r)$
     $\forall t \in \underset{R_i}{\pi}(m_\rho(r)) \implies \exists t' \in m_\rho(r) | t'[R_i] = t \implies \exists t_i \in r | t_i[R_i] = t'[R_i] \implies$
     $t = t'[R_i] = t_i[R_i] \in \underset{R_i}{\pi}(r)$
3. $m_\rho(m_\rho(r)) = \underset{R_1}{\pi}(m_\rho(r)) \bowtie \underset{R_2}{\pi}(m_\rho(r)) \bowtie ... \bowtie \underset{R_k}{\pi}(m_\rho(r)) = \underset{R_1}{\pi}(r) \bowtie \underset{R_2}{\pi}(r) \bowtie ... \bowtie \underset{R_k}{\pi}(r) = m_\rho(r)$

4.7.1 Controllare un join senza perdite

Per controllare se una decomposizione ha un join senza perdite si usa una serie di passaggi:

1. Si costruisce una tabella con $|R|$ numero di colonne e k righe
2. In ogni casella della tabella con colonna j e riga i scriviamo $\begin{cases} a & A_j \in R_i \\ b_i & A_j \notin R_i \end{cases}$
3. Per ogni dipendenza $(X \to Y) \in F$ se ci sono due tuple $t_1[X] = t_2[X]$ ma $t_1[Y] \neq t_2[Y]$ allora se una delle due è uguale ad a allora cambia anche l'altra in a mentre se nessuno dei due è uguale a a allora cambia uno delle due facendola diventare uguale all'altra.

4. Se in qualunque momento una riga avesse tutte a allora possiamo fermarci e sapere che ha un join senza perdite

5. Alla fine del controllo di tutte le dipendenze se nessuna riga ha tutte a allora ripartiamo a controllare su tutte le dipendenze, fino a quando un'intero ciclo di dipendenze non cambia nulla sulla tabella

I passaggi precedenti sono derivati da un algoritmo: **Dimostrazione per assurdo:**

Algoritmo: Calcolo Join senza perdite

Input:

- R: schema
- F: insieme di dipendenze
- ρ: decomposizione

```
def JoinLossless(R,F,ρ):
    r=Tabella
    changed=True
    while changed and ∄t in r|t = {a...a} :
        changed=False
        for (X → Y) in F :
            if t1[X]==t2[Y] and t1[X]!=t2[Y] :
                changed=True
                for A in Y :
                    if t1[A]==a :
                        t2[A]=a
                    else
                        t1[A]=t2[A]
    if ∃t in r|t = {a...a} :
        return True
    return False
```

Supponiamo che ρ abbia un join senza perdita $m_\rho(r) = r$ ma r non contenga nessuna tupla con solamente a.
Visto che non succede mai che $a \to b_i$ allora $\exists t_i \in \underset{R_i}{\pi}(r) | t_i = "a...a" \forall i$, precisamente nella riga corrispondente al sottoschema R_i.
Quindi $m_\rho(r)$ contiene una riga con tutte a.

4.8 Copertura minimale

Definizione di copertura minimale

Dato un insieme di dipendenze funzionali F, una copertura minimale di F è un insieme di dipendenze funzionali G tale che:

1. Tutte le dipendenze in G hanno un attributo singolo come dipendente (a destra)
2. $\forall(X \to Y) \in G \nexists X' \subset X | G \equiv [G - \{(X \to Y)\}] \cup \{(X' \to Y)\}$
3. $\forall(X \to Y) \in G \implies G \not\equiv G \backslash \{(X \to Y)\}$

Preso un insieme F possiamo calcolare la copertura minimale tramite 3 passaggi:

1. Con la regola di decomposizione divido tutti i dipendenti ad attributi singoli
2. Controllo se le dipendenze possono essere ridotte, cioè $\exists X' \subset X | (X' \to Y)$, in caso si possa sostituisco la dipendenze $(X \to Y)$ con $(X' \to Y)$, per farlo devo calcolare la chiusura di tutti i sottoinsiemi di X e controllare se contengono Y
3. Controllo per ogni dipendenza se $(X)^+_{F \backslash \{(X \to Y)\}}$ (X^+ calcolato sull'insieme F escludendo la dipendenza stessa)contiene Y, se si elimino la dipendenza $(X \to Y)$

4.9 Algoritmo di decomposizione

Dato uno schema relazionale R con insieme di dipendenze funzionali F esiste sempre una decomposizione ρ tale che:

1. Ogni sottoschema è in 3NF
2. Preserva F
3. Ha un join senza perdita

Per trovare la decomposizione possiamo usare un algoritmo composto da 3 passaggi:

1. Aggiungere a ρ un sottoschema S con tutti gli attributi che non appaiono in nessuna dipendenza
2. Tolgo questi attributi a R e controllo se c'è una dipendenza che contiene tutti gli attributi rimasti in $R - S$, se c'è aggiungo tutti questi attributi come unico sottoschema a ρ
3. Se non c'è per ogni dipendenza aggiungo come sottoschema tutti gli attributi che la compongono
4. Per avere un join senza perdita un sottoschema deve contenere una chiave, se non c'è allora dobbiamo aggiungerne una.

I passaggi precedenti sono derivati da un algoritmo: **Dimostrazione:**

Algoritmo: Calcolo decomposizione

Input:

- R: schema
- F: copertura minimale

```
def CalcolaDecomposizione(R,F):
    S={}
    ρ = {}
    S={A|∀(X → Y) ∈ F, A ∈ X, A ≠ Y}
    if S≠ ∅ :
        R=R-S
        ρ = ρ ∪ {S}
    if ∃(X → Y) ∈ F|X ∪ Y == R :
        ρ = ρ ∪ {R}
    else
        for (X → Y) in F :
            ρ = ρ ∪ {X ∪ Y}
    return ρ
```

1. Preserva F
   $G = \bigcup_{i=1}^{k} \pi_{R_i}(F)$
   $\forall (X \to Y) \in F \implies XY \in \rho \implies (X \to Y) \in G \implies$
   $F \equiv G$

2. I sottoschemi sono in 3NF
   Dividiamo i 3 casi possibili:

   - $S \in \rho \implies$ tutti gli attributi fanno parte di una chiave quindi è in 3NF
   - $R \in \rho \implies$ questa dipendenza ha forma $(R - A) \to A \implies R - A$ è la chiave, quindi $\forall (Y \to B) \in F \implies \begin{cases} B \neq A \implies B \in R - A \implies B \text{ primo} \\ B = A \implies Y = R - A \implies Y \text{ superchiave} \end{cases}$
   - $XA \in \rho \implies$ essendo F copertura minimale $\implies X$ è chiave di $XA \implies$ $\forall (Y \to B) \in F | YB \subseteq XA \implies \begin{cases} B = A \implies Y = X \implies Y \text{ superchiave} \\ B \neq A \implies B \in X \implies B \text{ primo} \end{cases}$

5

Organizzazione fisica