Appunti sulla Normalizzazione dei Database

Basato sulle slide del Prof. Danilo Montesi

17 maggio 2025

Indice

1 Normalizzazione nel Contesto dei Database

La **normalizzazione** è un processo fondamentale nella progettazione di database relazionali. Il suo scopo principale è organizzare i dati in modo da:

1. **Ridurre la ridondanza:** Evitare di ripetere le stesse informazioni in più punti.
2. **Eliminare le anomalie:** Prevenire problemi che possono sorgere durante l'inserimento, l'aggiornamento o la cancellazione dei dati.
3. **Garantire la qualità e l'integrità dei dati:** Assicurare che i dati siano coerenti e affidabili.

Le **Forme Normali (FN)** sono un insieme di regole che definiscono quanto "ben formata" è una tabella (relazione). Se una relazione non è in una forma normale adeguata, può presentare:

- **Ridondanze:** Dati duplicati inutilmente.
- **Comportamenti indesiderati durante gli aggiornamenti:** Ad esempio, la necessità di modificare lo stesso dato in più righe, con il rischio di dimenticarne qualcuna e creare inconsistenza.

La normalizzazione è una **tecnica di verifica** del design del database, non una metodologia di progettazione da zero. Prima progetti lo schema (magari con un modello E-R), poi lo verifichi e lo affini con la normalizzazione.

1.1 Esempio di Tabella con Anomalie

Consideriamo una tabella che traccia impiegati, progetti a cui lavorano, i loro stipendi, il budget dei progetti e il loro ruolo nel progetto:

Employee	Wage	Project	Budget	Role
Jones	20	Mars	2	Technician
Smith	35	Jupiter	15	Designer
Smith	35	Venus	15	Designer
Williams	55	Venus	15	Chief
Williams	55	Jupiter	15	Consultant
Williams	55	Mars	2	Consultant
Brown	48	Mars	2	Chief
Brown	48	Venus	15	Designer
White	48	Venus	15	Designer
White	48	Jupiter	15	Director

Questa tabella presenta diversi problemi (anomalie):

1. **Ridondanza:**
   - Lo stipendio (`Wage`) di un impiegato (es. Smith, 35) è ripetuto per ogni progetto a cui lavora.
   - Il budget (`Budget`) di un progetto (es. Jupiter, 15) è ripetuto per ogni impiegato che ci lavora.
2. **Anomalia di Aggiornamento (Update Anomaly):**
   - Se lo stipendio di Smith cambia, dobbiamo aggiornarlo in *tutte* le righe in cui Smith compare. Se ne dimentichiamo una, il database diventa inconsistente.
3. **Anomalia di Cancellazione (Deletion Anomaly):**
   - Se Jones smette di lavorare al progetto Mars (e Mars era il suo unico progetto), cancellando quella riga potremmo perdere l'informazione che Jones ha uno stipendio di 20 (se non ci sono altre tabelle che lo tracciano).
   - Similmente, se il progetto Mars viene cancellato e Jones e Brown lavoravano solo a Mars, perderemmo le informazioni su Jones e Brown.
4. **Anomalia di Inserimento (Insertion Anomaly):**

- Non possiamo inserire un nuovo impiegato con il suo stipendio se non è ancora assegnato a un progetto.
- Non possiamo inserire un nuovo progetto con il suo budget se nessun impiegato ci sta ancora lavorando.

1.2 Perché questa situazione è indesiderabile?

Perché stiamo mescolando diversi "concetti" o "pezzi di informazione" nella stessa tabella:

- Informazioni sugli impiegati e i loro stipendi.
- Informazioni sui progetti e i loro budget.
- Informazioni sul ruolo di un impiegato *all'interno di uno specifico progetto*.

2 Dipendenze Funzionali (Functional Dependencies - FD)

Per studiare e risolvere queste anomalie in modo sistematico, introduciamo il concetto di **Dipendenza Funzionale (FD)**. Una FD è un vincolo di integrità che descrive una relazione tra attributi all'interno di una tabella.

2.1 Definizione Formale

Data una relazione r con uno schema $R(X)$ (dove X è l'insieme di tutti gli attributi), e dati due sottoinsiemi non vuoti di attributi Y e Z (contenuti in X), esiste una dipendenza funzionale $Y \to Z$ (si legge "Y determina funzionalmente Z" o "Z dipende funzionalmente da Y") se e solo se: *Per ogni coppia di tuple (righe)* t_1 *e* t_2 *in* r*, se i valori degli attributi in* Y *sono uguali in* t_1 *e* t_2 *(cioè* $t_1[Y] = t_2[Y]$*), allora anche i valori degli attributi in* Z *devono essere uguali (cioè* $t_1[Z] = t_2[Z]$*).*

In parole povere: se conosci il valore di Y, puoi determinare *univocamente* il valore di Z.

2.2 Spiegazione in termini più semplici

"Determinare funzionalmente" significa semplicemente che se conosci il valore di un attributo, puoi conoscere con certezza il valore di un altro attributo.

2.2.1 Esempi dalla tabella precedente

- Employee $\to$ Wage
- Project $\to$ Budget
- $\{$Employee, Project$\}$ $\to$ Role

2.2.2 Spiegazione

- Employee $\to$ Wage significa: "Se sai chi è l'impiegato, sai sicuramente quanto guadagna". Nella nostra tabella, ogni volta che appare "Smith", il salario è sempre "35", ogni volta che appare "Jones" il salario è sempre "20". Il nome dell'impiegato *determina* univocamente il suo stipendio.
- Project $\to$ Budget significa: "Se sai qual è il progetto, sai sicuramente qual è il suo budget". Ogni volta che vedi "Mars" come progetto, il budget è sempre "2", ogni volta che vedi "Jupiter", il budget è sempre "15".

2.3 FD Triviali e Non Triviali

- Una FD $Y \to A$ è **triviale** se $A \subseteq Y$ (es. $\{$Employee, Project$\}$ $\to$ Project). Sono sempre vere e poco utili.
- Una FD $Y \to A$ è **non triviale** se $A \not\subseteq Y$. Sono queste che ci interessano per la normalizzazione.

2.3.1 Spiegazione semplice con esempi pratici

Dipendenza Funzionale Triviale - In termini semplici, è come dire "se conosci qualcosa, allora conosci anche una parte di quel qualcosa".

- **Esempio pratico:** In una tabella SQL `Clienti(ID, Nome, Cognome, Email)`, la dipendenza funzionale `{ID, Nome, Cognome} → Nome` è triviale perché ovviamente se conosci l'insieme {ID, Nome, Cognome}, allora conosci anche il Nome (che è già incluso nell'insieme).
- **In SQL:** Se scrivi `SELECT Nome FROM Clienti WHERE ID = 123 AND Nome = 'Mario' AND Cognome = 'Rossi'`, è ovvio che otterrai 'Mario' come risultato, perché è nelle condizioni stesse della query.

Dipendenza Funzionale Non Triviale - Significa che conoscendo alcuni attributi, puoi determinare altri attributi che *non* sono già contenuti nei primi.

- **Esempio pratico:** Nella tabella `Clienti(ID, Nome, Cognome, Email)`, la dipendenza `ID → Nome` è non triviale perché il Nome non è parte dell'ID e non è ovvio che l'ID determini il Nome.
- **In SQL:** Se scrivi `SELECT Nome FROM Clienti WHERE ID = 123`, otterrai un nome specifico perché esiste una dipendenza funzionale dall'ID al Nome (supponendo che ID sia una chiave primaria).

Perché le FD triviali non sono utili per la normalizzazione? Perché non rivelano nulla di nuovo sulla struttura dei dati. Sono sempre vere per definizione e non causano anomalie. Le dipendenze non triviali, invece, possono causare anomalie se non trattate correttamente.

2.4 Come le FD causano anomalie

Le anomalie sorgono principalmente quando abbiamo FD $X \to Y$ dove X **non è una superchiave** (o chiave candidata) della tabella.

- Employee $\to$ Wage: `Employee` da solo non è la chiave. Causa ridondanza.
- Project $\to$ Budget: `Project` da solo non è la chiave. Causa ridondanza.
- $\{$Employee, Project$\} \to$ Role: $\{$Employee, Project$\}$ è (probabilmente) la chiave primaria. Questa FD **non causa anomalie**.

Le anomalie sono quindi causate dalla presenza di informazioni eterogenee.

3 Forma Normale di Boyce-Codd (BCNF)

La BCNF è una delle forme normali più stringenti e desiderabili.

3.1 Definizione

Una relazione r è in **BCNF** se, per ogni dipendenza funzionale non triviale $X \to Y$ definita su r:

- X è una **superchiave** di r.

Nella nostra tabella di esempio iniziale, non è in BCNF a causa di Employee $\to$ Wage e Project $\to$ Budget.

3.1.1 Spiegazione della violazione BCNF

Ricordiamo la definizione di BCNF: per ogni dipendenza funzionale non triviale $X \to Y$, X deve essere una superchiave della relazione.

Nel nostro esempio:

- Employee $\to$ Wage: L'attributo `Employee` determina funzionalmente `Wage`. Ma `Employee` da solo non è una superchiave della tabella, perché non può determinare univocamente tutti gli altri attributi (come `Project`, `Budget`, `Role`). La chiave primaria della tabella è $\{$Employee, Project$\}$.

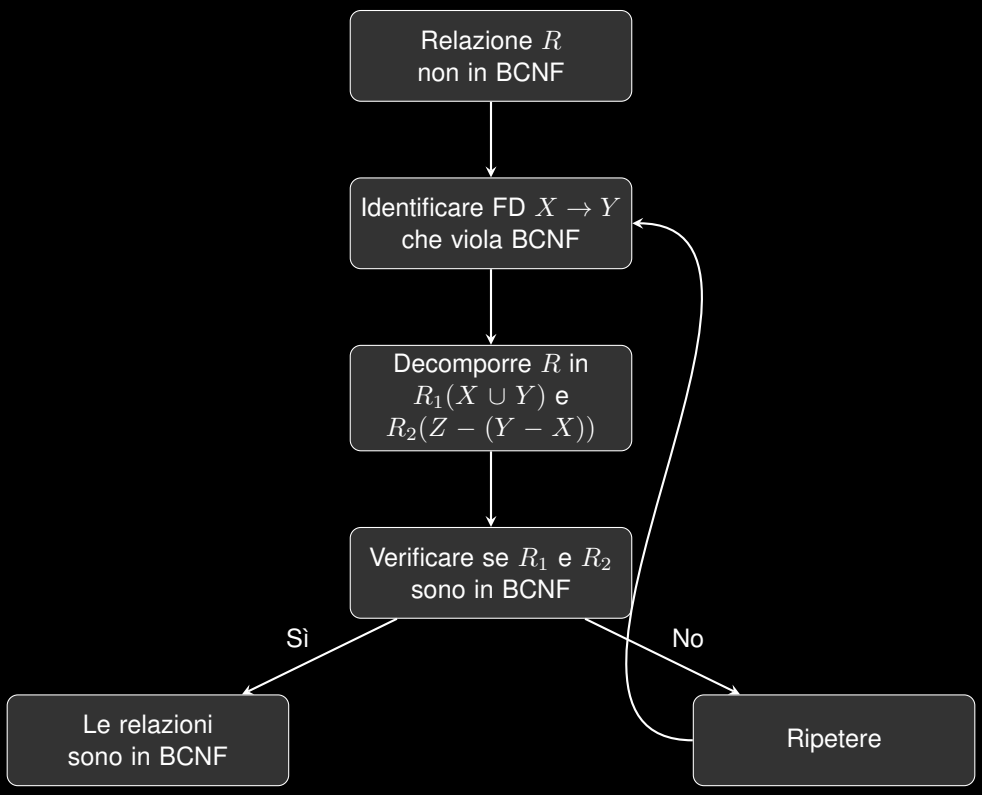
- Project $\rightarrow$ Budget: Similmente, `Project` determina `Budget`, ma `Project` da solo non è una superchiave della tabella.

Queste violazioni causano le anomalie di inserimento, cancellazione e aggiornamento che abbiamo descritto in precedenza.

Perché questo viola BCNF? Perché BCNF richiede che quando un attributo (o gruppo di attributi) determina funzionalmente un altro attributo, il primo deve essere una "superchiave". In termini semplici, una superchiave è un attributo (o gruppo di attributi) che può identificare univocamente ogni riga nella tabella.

3.2 Cosa fare se una relazione non è in BCNF?

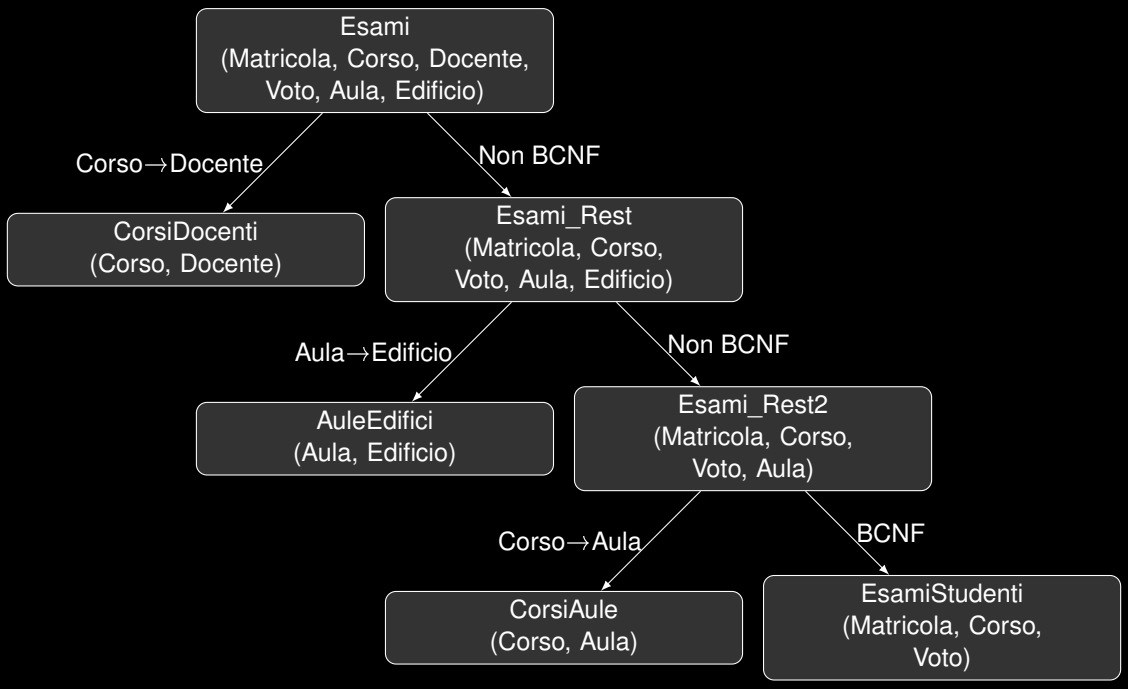
Si **decompone** la relazione in più relazioni più piccole, ognuna delle quali sia in BCNF.



3.2.1 Esempio pratico di decomposizione

```
Esami(Matricola, Corso, Docente, Voto, Aula, Edificio)
    ↓ FDs che violano BCNF:
    * Corso → Docente     (Corso non è superchiave)
    * Aula → Edificio     (Aula non è superchiave)
```

Decomposizione in BCNF:



3.2.2 Esempio con dipendenze più complesse

`OrdiniFornitori(IDOrdine, CodiceArticolo, QuantitàOrdinata,`
`CodiceFornitore, RagioneSocialeFornitore, CittàFornitore)`

`FDs:`

- `IDOrdine → CodiceFornitore`
- `{IDOrdine, CodiceArticolo} → QuantitàOrdinata` (chiave)
- `CodiceFornitore → {RagioneSocialeFornitore, CittàFornitore}`

Ordini	**Fornitori**	**DettagliOrdine**
IDOrdine	CodiceFornitore	IDOrdine, CodiceArticolo
CodiceFornitore	RagioneSocialeFornitore	QuantitàOrdinata
	CittàFornitore	

3.3 Esempio di Decomposizione (per la tabella iniziale)

1. **ImpiegatiStipendi**(Employee, Wage)
2. **ProgettiBudget**(Project, Budget)
3. **ImpiegatiRuoliProgetto**(Employee, Project, Role)

Questa decomposizione elimina le anomalie.

3.4 Qualità della Decomposizione

Quando decomponiamo una tabella, dobbiamo assicurarci due proprietà fondamentali:

3.4.1 Lossless Join Property (Proprietà di Join Senza Perdita)

Dobbiamo essere in grado di ricreare la tabella originale facendo il JOIN delle tabelle decomposte. **Condizione:** Una decomposizione di $r(X)$ in $r_1(X_1)$ e $r_2(X_2)$ è senza perdita se l'intersezione degli attributi $X_0 = X_1 \cap X_2$ forma una chiave per almeno una delle relazioni decomposte ($X_0 \to X_1$ oppure $X_0 \to X_2$).

Esempio di Decomposizione CON PERDITA: Supponiamo R(Employee, Project, Office) con FD: Employee $\to$ Office e Project $\to$ Office. Decomposizione in:

- R_1(Employee, Office)
- R_2(Project, Office)

Tabella originale:

Employee	Project	Office
Smith	Alpha	A101
Jones	Beta	B202
Brown	Alpha	C303

Tabelle decomposte:

Employee	Office
Smith	A101
Jones	B202
Brown	C303

Project	Office
Alpha	A101
Beta	B202
Alpha	C303

Risultato del JOIN (genera tuple spurie):

Employee	Project	Office	Tuple Spurie
Smith	Alpha	A101	
Jones	Beta	B202	
Brown	Alpha	C303	
Smith	Alpha	C303	✓
Brown	Alpha	A101	✓

Attributo comune: `Office`. `Office` non è chiave né per R_1 né per R_2. Il join può generare tuple spurie.

Esempio di Decomposizione SENZA PERDITA: Tabella R(Employee, Project, Office) con FD: Employee $\rightarrow$ Office e chiave primaria {Employee, Project}. Decomposizione in:

- R_1(Employee, Office)
- R_2(Employee, Project)

Tabella originale:

Employee	Project	Office
Smith	Alpha	A101
Smith	Beta	A101
Jones	Gamma	B202

Tabelle decomposte:

Employee	Office
Smith	A101
Jones	B202

Employee	Project
Smith	Alpha
Smith	Beta
Jones	Gamma

Risultato del JOIN (senza tuple spurie):

Employee	Project	Office
Smith	Alpha	A101
Smith	Beta	A101
Jones	Gamma	B202

Attributo comune: `Employee`. `Employee` è chiave per R_1. Lossless.

3.4.2 Dependency Preservation (Conservazione delle Dipendenze)

Tutte le dipendenze funzionali originali devono poter essere verificate esaminando una singola tabella nello schema decomposto.

In parole semplici: La proprietà di conservazione delle dipendenze garantisce che, dopo aver decomposto una tabella in più tabelle, ogni regola (dipendenza funzionale) della tabella originale possa essere verificata esaminando *una sola* delle tabelle risultanti, senza dover eseguire join.

Se F è l'insieme delle dipendenze funzionali su R, e R viene decomposto in $R_1, R_2, \ldots, R_n$, allora:

- Per ogni dipendenza $X \to Y$ in F, deve esistere almeno un R_i tale che $X \cup Y \subseteq R_i$
- Se nessun R_i contiene tutti gli attributi di una dipendenza, allora quella dipendenza non può essere verificata senza combinare più tabelle

Perché è importante? Senza la conservazione delle dipendenze:

- Diventa difficile mantenere l'integrità dei dati
- Le operazioni di controllo richiedono join costosi
- L'efficienza del database ne risente significativamente

Esempio di Decomposizione che **NON** preserva le dipendenze:

$R(\text{Employee, Project, Office})$ con dipendenze:

- Employee $\to$ Office
- Project $\to$ Office

Tabella originale R:

Employee	**Project**	**Office**
Jones	Mars	Rome
Smith	Jupiter	Milan
Smith	Venus	Milan
White	Saturn	Milan
White	Venus	Milan
White	Mars	Milan

Decomposizione in $R_1(\text{Employee, Office})$ e $R_2(\text{Employee, Project})$:

Employee	**Office**
Jones	Rome
Smith	Milan
White	Milan

Employee	**Project**
Jones	Mars
Smith	Jupiter
Smith	Venus
White	Saturn
White	Venus
White	Mars

La FD Project $\to$ Office **non è preservata** perché:

- In R_1 manca l'attributo Project, quindi non può verificare Project $\to$ Office
- In R_2 manca l'attributo Office, quindi non può verificare Project $\to$ Office
- Non esiste nessuna singola tabella in cui possiamo verificare questa dipendenza

Una decomposizione alternativa che preserva le dipendenze:

Per preservare tutte le dipendenze funzionali, possiamo decomporre R in tre tabelle:

- $R_1(\text{Employee, Office})$ - preserva Employee $\to$ Office
- $R_2(\text{Project, Office})$ - preserva Project $\to$ Office
- $R_3(\text{Employee, Project})$ - mantiene la relazione tra Employee e Project

Decomposizione che preserva le dipendenze:

Employee	Office
Jones	Rome
Smith	Milan
White	Milan

Project	Office
Mars	Rome
Jupiter	Milan
Venus	Milan
Saturn	Milan

Employee	Project
Jones	Mars
Smith	Jupiter
Smith	Venus
White	Saturn
White	Venus
White	Mars

Questa decomposizione preserva tutte le dipendenze, (perché ogni dipendenza funzionale $X \to Y$ è contenuta interamente in almeno una delle tabelle risultanti: Employee $\to$ Office è in R_1 e Project $\to$ Office è in R_2) ma purtroppo **non garantisce** la proprietà di join senza perdita (lossless join). Infatti, quando riuniamo le tre tabelle, potremmo generare tuple spurie (ad esempio, dal join otterremmo che White lavora a Mars con Office = Rome, mentre nella tabella originale White lavora a Mars ma con Office = Milan; oppure potremmo ottenere Smith lavora a Mars, che non esiste nella relazione originale).
Il trade-off tra "Dependency Preservation" e "Lossless Join" è uno dei motivi per cui è stata definita la Terza Forma Normale (3NF).

4 Recap: Quello che abbiamo imparato finora

Piccolo riassunto di quanto visto finora:

- **Normalizzazione:** È un processo per organizzare i dati in un database in modo da evitare ridondanze e anomalie. Serve a verificare e migliorare uno schema già progettato, non a crearlo da zero.
- **Anomalie:** Sono problemi che possono verificarsi quando lo schema del database non è ben progettato:
  - **Anomalia di inserimento:** Non posso inserire certi dati se non ho altri dati correlati.
  - **Anomalia di cancellazione:** Cancellando alcuni dati perdo accidentalmente altre informazioni importanti.
  - **Anomalia di aggiornamento:** Devo aggiornare lo stesso dato in più punti, rischiando inconsistenze.
- **Dipendenze Funzionali (FD):** Sono vincoli che esprimono come un attributo (o set di attributi) determina univocamente un altro attributo. Esempio: `Codice_Fiscale → Data_Nascita, Comune_Nascita` significa che conoscendo il CF posso determinare con certezza la data di nascita e il comune di nascita.
- **Forma Normale di Boyce-Codd (BCNF):** Una tabella è in BCNF se per ogni dipendenza funzionale X $\to$ Y, X deve essere una superchiave (cioè deve poter identificare univocamente ogni riga della tabella).
- **Decomposizione:** Quando una tabella non è in BCNF, la dividiamo in tabelle più piccole che siano in BCNF.
- **Proprietà della decomposizione:**
  - **Lossless Join:** La decomposizione deve permettere di ricostruire la tabella originale senza generare tuple spurie. Garantita se l'intersezione degli attributi forma una chiave per almeno una delle tabelle.
    - **Esempio:** Se decompongo `Studenti(Matricola, Nome, CorsoDiLaurea)` in `Anagrafica(Matricola, Nome)` e `Iscrizione(Matricola, CorsoDiLaurea)`, l'attributo comune `Matricola` è chiave per entrambe, quindi il join sarà senza perdita.
  - **Dependency Preservation:** Le dipendenze funzionali originali devono essere verificabili nelle tabelle decomposte senza fare join.
    - **Esempio:** Se ho la FD `Matricola → CorsoDiLaurea` nella tabella originale, dopo la decomposizione devo poterla verificare in una singola tabella, ovvero `Iscrizione`.
- **Il problema:** Non sempre è possibile ottenere una decomposizione che sia sia lossless che preservi tutte le dipendenze e sia in BCNF.

Questo è il motivo per cui ora introduciamo la Terza Forma Normale (3NF), che è un po' meno restrittiva della BCNF ma garantisce sempre una decomposizione che preserva le dipendenze e ha la proprietà di lossless join.

5 Terza Forma Normale (3NF)

La 3NF è una forma normale leggermente meno stringente della BCNF che permette sempre una decomposizione lossless e che preserva le dipendenze.

5.1 Definizione

Una relazione r è in **3NF** se, per ogni dipendenza funzionale non triviale $X \to Y$ definita su r, almeno una delle seguenti condizioni è vera:

1. X è una **superchiave** di r (condizione BCNF). **OPPURE**
2. Ogni attributo in Y è parte di **almeno una chiave candidata** di r (cioè, ogni attributo in Y è un "attributo primo").

5.2 BCNF vs 3NF

- BCNF è più restrittiva. Ogni relazione in BCNF è anche in 3NF.
- Una relazione in 3NF potrebbe non essere in BCNF.
- Si può sempre decomporre una relazione in 3NF in modo lossless e preservando le dipendenze.
- Se una relazione ha una sola chiave candidata, allora 3NF e BCNF sono equivalenti.

5.3 Esempio 3NF (ma non BCNF)

Consideriamo una tabella $R(\text{Chief, Project, Office})$. Supponiamo di avere le seguenti Dipendenze Funzionali (FDs):

1. $\{\text{Project, Office}\} \to \text{Chief}$ (questa è una chiave candidata)
2. $\text{Chief} \to \text{Office}$

Un'istanza di esempio della tabella R potrebbe essere:

Chief	**Project**	**Office**
Rossi	Alpha	Stanza101
Rossi	Beta	Stanza101
Verdi	Gamma	Stanza202
Bianchi	Alpha	Stanza303

Da questa tabella, osserviamo:

- Per la FD $\{\text{Project, Office}\} \to \text{Chief}$:
  - $(\text{Alpha, Stanza101}) \to \text{Rossi}$
  - $(\text{Beta, Stanza101}) \to \text{Rossi}$
  - $(\text{Gamma, Stanza202}) \to \text{Verdi}$
  - $(\text{Alpha, Stanza303}) \to \text{Bianchi}$

  La coppia $\{\text{Project, Office}\}$ identifica univocamente `Chief`, quindi è una chiave candidata.
- Per la FD $\text{Chief} \to \text{Office}$:
  - $\text{Rossi} \to \text{Stanza101}$

- Verdi $\rightarrow$ Stanza202
- Bianchi $\rightarrow$ Stanza303

L'attributo `Chief` determina univocamente `Office`.

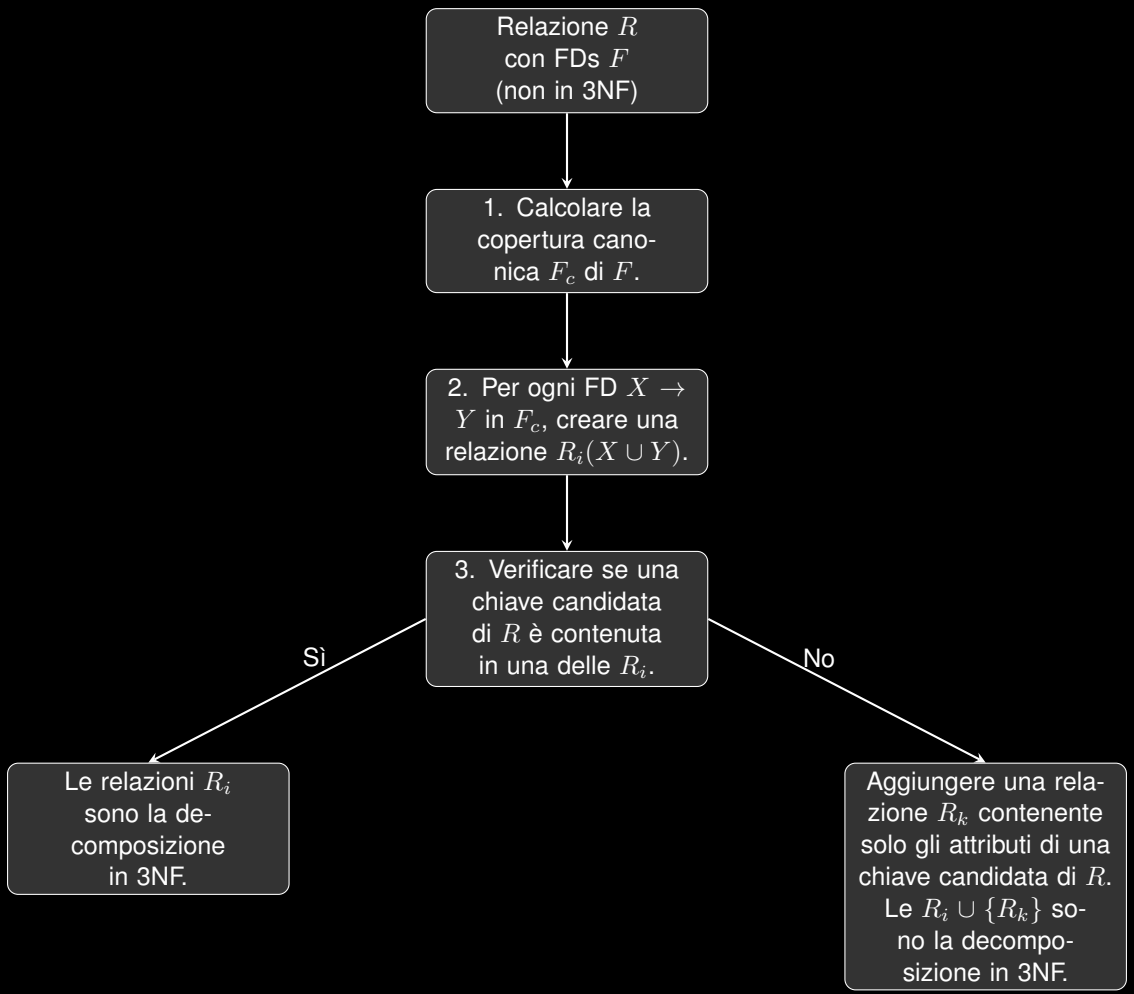
Analizziamo ora la FD Chief $\rightarrow$ Office rispetto alle forme normali:

- **BCNF:** La FD Chief $\rightarrow$ Office viola la BCNF perché Chief non è una superchiave. (Ad esempio, `Chief` da solo non determina `Project`: Rossi è associato sia al progetto Alpha che Beta). **Quindi, R NON è in BCNF.**
- **3NF:** Per la FD Chief $\rightarrow$ Office:
  1. Chief non è una superchiave. (Condizione 1 non soddisfatta)
  2. MA, ogni attributo in Y (che è $\{$Office$\}$ in questo caso) è parte di almeno una chiave candidata di R. L'attributo Office è infatti parte della chiave candidata $\{$Project, Office$\}$. (Condizione 2 soddisfatta)

  Poiché la seconda condizione è soddisfatta, **la relazione R È in 3NF.**

5.4 Algoritmo di Decomposizione in 3NF (Idea Generale)

1. Trova un insieme minimo di dipendenze funzionali (copertura canonica).
2. Per ogni FD $X \rightarrow Y$ in questa copertura, crea una tabella con attributi $X \cup Y$.
3. Se nessuna delle tabelle create contiene una chiave candidata della relazione originale, aggiungi un'ulteriore tabella contenente solo gli attributi di una chiave candidata originale.



5.5 Approccio Pratico Consigliato

1. Decomponi la relazione per raggiungere la 3NF.
2. Verifica se le tabelle risultanti sono anche in BCNF.
3. Se una tabella è in 3NF ma non in BCNF, valuta il trade-off.

5.6 Teoria delle Dipendenze e Implicazioni

Dato un insieme di dipendenze funzionali F, possiamo derivare altre dipendenze funzionali. Diciamo che F implica f se ogni relazione che soddisfa F soddisfa anche f. L'insieme di tutte le dipendenze implicite da F è chiamato **chiusura di F (F^+)**.

5.6.1 Assiomi di Armstrong

1. **Riflessività:** Se $Y \subseteq X$, allora $X \to Y$.
2. **Aumento:** Se $X \to Y$, allora $XZ \to YZ$.
3. **Transitività:** Se $X \to Y$ e $Y \to Z$, allora $X \to Z$.
    - Esempio: MatricolaStudente $\to$ CodiceCorsoLaurea e CodiceCorsoLaurea $\to$ NomeCorsoLaurea. Allora, MatricolaStudente $\to$ NomeCorsoLaurea.

5.6.2 Chiusura di un insieme di attributi X^+

L'insieme di tutti gli attributi che sono funzionalmente determinati da X, dato un insieme di FDs F.

5.6.3 Copertura Minima (o Canonica)

Un insieme "minimale" di FDs equivalente a F, senza ridondanze.

6 Normalizzazione nel Design Concettuale (Modello E-R)

La teoria della normalizzazione può essere usata anche per verificare la qualità di un modello Entità-Relazione.

6.1 Esempio: Normalizzazione su Entità

Considera un'entità `Prodotto` con attributi: `Codice` (PK), `NomeProdotto`, `Prezzo`, `PartitaIVAFornitore`, `NomeFornitore`, `IndirizzoFornitore`.

Identifichiamo una FD: PartitaIVAFornitore $\to$ NomeFornitore, IndirizzoFornitore. Qui, `PartitaIVAFornitore` non è la chiave di `Prodotto`. Questo viola le forme normali.

Decomposizione dell'Entità:

- Entità `Prodotto`(`Codice`, `NomeProdotto`, `Prezzo`)
- Entità `Fornitore`(`PartitaIVAFornitore`, `NomeFornitore`, `IndirizzoFornitore`)
- Relazione `Fornisce` tra `Fornitore` e `Prodotto`.

Nello schema proposto dalla slide 54:

- `Product(Code, Name, Price)`
- `Supplier(VATNum, Name, Address)`
- `Supply` (relationship)

6.2 Esempio: Normalizzazione su Relazioni (Relationship)

Considera una relazione `Tesi` che collega `Studente`, `Professore`, `DipartimentoProf`, `CorsoLaureaStudente`. Assumiamo che (MatricolaStudente, IDProfessore) sia la chiave. FDs:

- MatricolaStudente $\rightarrow$ CorsoLaureaStudente
- IDProfessore $\rightarrow$ DipartimentoProf

Nella tabella `Tesi`(MatricolaStudente, IDProfessore, CorsoLaureaStudente, DipartimentoProf):

- IDProfessore $\rightarrow$ DipartimentoProf: `IDProfessore` è parte della chiave, non la chiave intera. `DipartimentoProf` non è un attributo primo. Viola 3NF/BCNF.

Decomposizione della Relazione (Conceptual Level):

1. Creare un'entità `Professore` con attributo `Dipartimento` (slide 50: `Professor -(1,1)- Work -(0,N)- Dept`).
2. Creare un'entità `Studente` con attributo `CorsoLaurea` (slide 52: `Student -(1,1)- Enroll -(0,N)- Degree`).
3. La relazione `Tesi` ora collegherebbe solo `Studente` e `Professore` (slide 52: `Professor -(0,N)- Thesis -(0,1)- Student`).

Questo processo porta a un modello concettuale più robusto.