
# UNIVERSITÀ DI PISA

**Dipartimento di Informatica**
**Corso di Laurea Triennale in Informatica**

Corso 3° anno - 6 CFU

## Basi di dati

**Professore:**
Prof. Riccardo Guidotti

**Autore:**
Filippo Ghirardini

---

**Anno Accademico 2024/2025**

# Contents

# Basi di dati

Realizzato da: Ghirardini Filippo

A.A. 2024-2025

---

# 1 Introduzione

**Definizione 1.0.1** (Base di dati)**.** *Insieme* ***strutturato*** *e* ***organizzato*** *di dati omogenei utilizzati per il supporto allo svolgimento di attività (ente, azienda, ufficio, persona).*

**Definizione 1.0.2** (Database Management System)**.** *Sistema software progettato per consentire la* ***creazione****,* ***manipolazione*** *e* ***interrogazione*** *di uno o più database in modo corretto ed efficiente.*

Le figure coinvolte nella costruzione di una *base di dati* sono:

- **Committente**
  - Dirigente
  - Operatore
- **Fornitore**
  - Direttore del progetto
  - Analista
  - Progettista di BD
  - Programmatore di applicazioni che usano BD
- **Manutenzione** e messa a punto
  - Gestione del DBMS
  - Amministratore del DBMS

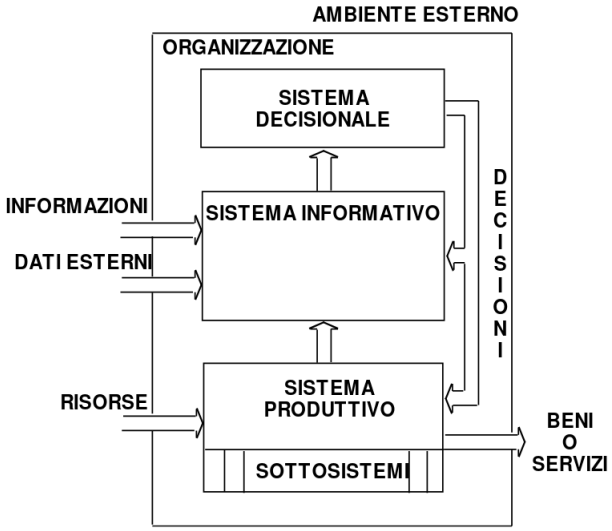
## 1.1 Sistema informativo

**Definizione 1.1.1** (Sistema informativo)**.** *Una combinazione di* ***risorse****,umane e materiali, e di procedure organizzate allo scopo di:*

- ***raccogliere***
- ***archiviare***
- ***elaborare***
- ***scambiare***

*le* ***informazioni*** *necessarie alle* ***attività****:*

- ***operative*** *(servizio)*
- *di* ***programmazione*** *e* ***controllo*** *(gestione)*
- *di* ***pianificazione strategica*** *(governo)*

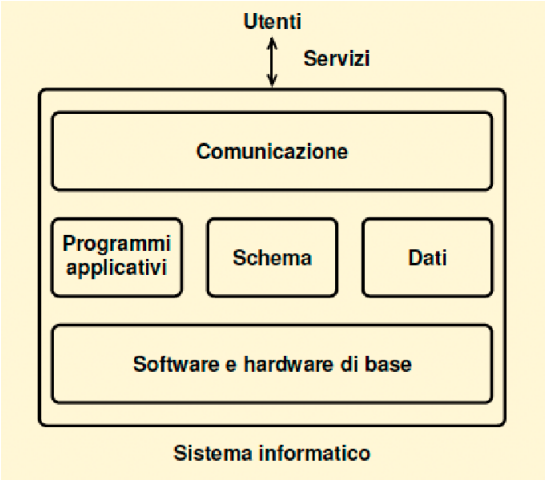


**Esempio 1.1.1** (Azienda manifatturiera)**.** Un'azienda manifatturiera avrà un sistema informativo che prevede la **gestione** di ordini (a clienti e fornitori), pagamenti e del magazzino, nonché la **pianificazione** e il controllo dei costi.

## 1.2 Sistema informatico

Nello specifico, si utilizza un **sistema informatico** per eseguire parte delle operazioni del sistema informativo in maniera **automatizzata**.

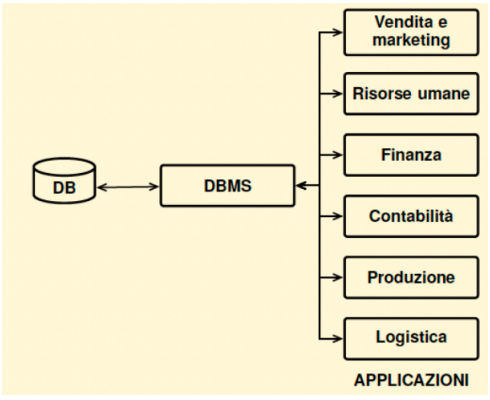
**Definizione 1.2.1** (Sistema informatico)**.** *Insieme delle tecnologie informatiche e della comunicazione (ICT) a supporto delle attività di un'organizzazione.*




### 1.2.1 Sistema informatico operativo

In questo tipo di sistemi i dati sono organizzati in BD (gestite a loro volta da DBMS) e le applicazioni sono usate per svolgere attività **strutturate** e **ripetitive** (e.g. amministrazione, vendite, produzione, HR), ovvero **On-Line Transaction Processing** (operazioni **semplici** e con **pochi dati**).

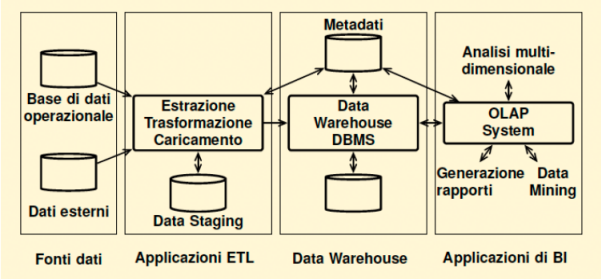
**Definizione 1.2.2** (Transaction Processing System)**.** *Sistema basato su transazioni: serie di azioni S che se non andate a buon fine lasciano la BD nello stesso stato in cui era prima che S iniziasse.*




### 1.2.2 Sistema informatico direzionale

In questo tipo di sistemi i dati sono organizzati in una **Data Warehouse** e gestiti da un opportuno sistema. A differenza di quello operazionale, dove i dati sono aggiornati tempestivamente, in questo l'aggiornamento avviene in maniera periodica. Inoltre l'uso principale è per **On-Line Analytical Processing**, ovvero analisi dei dati a supporto delle decisioni (operazioni complesse e con molti dati).

**Definizione 1.2.3** (Business Intelligence)**.** *Le applicazioni di Business intelligence sono strumenti di supporto ai processi di controllo delle prestazioni aziendali e di decisione manageriale.*




Di seguito una tabella che riassume le differenze tra **OLTP** e **OLAP**:

| | **OLTP** | **OLAP** |
|---|---|---|
| **Scopi** | Supporto operatività | Supporto decisioni |
| **Utenti** | Molti, esecutivi | Pochi, dirigenti e analisti |
| **Dati** | Pochi, analitici, relazionali | Molti, sintetici, multidimensionali |
| **Usi** | Noti a priori | Poco prevedibili |
| **Orientamento** | Applicazione | Soggetto |
| **Aggiornamento** | Frequente | Raro |
| **Visione dei dati** | Corrente | Storica |
| **Ottimizzati per** | Transazioni | Analisi |

### 1.2.3 Big Data

Con questo termine ci si riferisce alle situazioni in cui DB o DW sono troppo lenti o restrittivi a causa della natura dei dati: **Volume**, **Varietà** e **Velocità**. In questo caso si usano approcci come:

- NoSQL
- **Data mining**: fase di un processo interattivo e iterativo che cerca di estrarre modelli utili per prendere decisioni da un insieme di dati. Il modello sarà una rappresentazione concettuale che evidenzia alcune caratteristiche implicite
- Machine learning
- Data Lake

## 1.3 Base di dati

**Definizione 1.3.1** (Base di dati)**.** *Una BD è una raccolta di* ***dati permanenti****, gestiti da un elaboratore elettronico, suddivisi in:*

- ***Metadati*** *o schema: definizioni che* ***descrivono*** *i dati, pongono* ***restrizioni*** *su di essi, indicano le possibili* ***relazioni*** *ed* ***operazioni***
- ***Dati****: rappresentazione di fatti conforme allo schema. Sono organizzati in insiemi* ***strutturati*** *ed* ***omogenei*** *fra i quali sono definite* ***relazioni****. Hanno le seguenti caratteristiche:*
  - ***Molti*** *rispetto ai metadati*
  - ***Permanenti*** *fino ad esplicita cancellazione*
  - *Accessibili tramite* ***transazioni*** *(unità atomiche che non possono avere effetti parziali)*
  - ***Protetti*** *da utenti ed errori*
  - *Utilizzabili in* ***parallelo***

### 1.3.1 Sistemi per BD

Il DBMS si occupa di garantire le caratteristiche della BD, controllandone i dati e gestendone l'accessibilità.

**Definizione 1.3.2** (DBMS)**.** *Un **Database Management System** è un sistema centralizzato o distribuito che offre opportuni linguaggi per:*

- *Definire lo **schema***
- *Scegliere le **strutture dati** per memorizzazione e accesso*
- ***Memorizzare**, **recuperare** e **modificare** i dati interattivamente o da programmi*

Il modello più diffuso è quello **relazionale** che utilizza come meccanismo principale di astrazione la **tabella**, ovvero un insieme di **record** con campi elementari (definiti assieme al nome dallo schema).
Le principali funzionalità del DBMS sono:

- **Definizione** e **uso** della BD
- **Controllo** dei dati
- **Amministrazione**: definizione e modifica degli schemi, controllo e messa a punto del sistema, gestione dei diritti di accesso, strumenti di ripristino
- **Sviluppo**: strumenti per creare applicazioni. E.g. per produrre grafici, rapporti o GUI

### 1.3.2 Linguaggi

Per **definire** e **usare** una BD esistono due tipi di linguaggi:

- **Data Definition Language**: per la definizione dello *schema*
- **Data Manipulation Language**: permette agli operatori di accedere ai dati e modificarli

Dato che gli utenti sono di diversi tipi un DBMS deve prevedere più modalità d'uso: **GUI**, linguaggio di **querying** per non esperti, linguaggio di **programmazione** per integrarsi con applicazioni e linguaggio per lo **sviluppo di interfacce**.
Un esempio di linguaggio interattivo è **SQL** mentre un linguaggio ad-hoc è **PL/SQL**.

#### 1.3.2.1 Livelli di descrizione

Ci sono tre livelli a cui descrivere i dati:

- **Logico**: descrive la struttura degli insiemi di dati e delle relazioni fra loro, secondo un certo modello dei dati, senza nessun riferimento alla loro organizzazione fisica nella memoria permanente
- **Fisico**: descrive come vanno organizzati fisicamente i dati nelle memorie permanenti e quali strutture dati ausiliarie prevedere per facilitarne l'uso
- **Esterno**: descrive come deve apparire la struttura della base di dati ad una certa applicazione

È necessario che ci sia **indipendenza** logica e fisica:

- **Logica**: i programmi applicativi non devono essere modificati in seguito a modifiche dello schema logico
- **Fisica**: i programmi applicativi non devono essere modificati in seguito a modifiche dell'organizzazione fisica dei dati

### 1.3.3 Controllo di BD

Una base di dati deve sempre garantire **integrità** (mantenimento delle proprietà specificate nello schema), **sicurezza** (chi e come può accedere ai dati) e **affidabilità**.

**1.3.3.1 Affidabilità** La BD deve garantire la protezione dei dati da malfunzionamenti HW, SW e da interferenze dovute ad accesso parallelo.

**Definizione 1.3.3** (Malfunzionamento)**.** *Evento a causa del quale la BD si può trovare in uno stato scorretto.*

Le transazioni permettono di garantire affidabilità.

**Definizione 1.3.4** (Transazione)**.** *Una sequenza di azioni di lettura e scrittura in memoria permanente e di elaborazioni di dati in memoria temporanea, con le seguenti proprietà:*

- ***Atomicità**: le transazioni che terminano prematuramente sono trattate dal sistema come se non fossero mai iniziate ed eventuali effetti sono annullati*
- ***Persistenza**: le modifiche di una transazione terminata normalmente non sono alterabili da eventuali malfunzionamenti*
- ***Serializzabilità**: nel caso di transazioni concorrenti l'effetto è quello di una esecuzione seriale*

In caso di malfunzionamento rilevato si procede con:

1. Interruzione della transazione o del sistema
2. Messa in atto di procedure di recupero

I tipi di malfunzionamento sono:

| Tipo | Perdita dati |
|---|---|
| Transaction | Nessuna |
| System | Memoria non permanente |
| Media | Memoria permanente |

### 1.3.4 Vantaggi e svantaggi

I DBMS garantiscono:

- **Indipendenza**, **integrità** e **sicurezza** dei dati
- Gestione degli accessi **concorrenti** e **interattiva** in maniera sicura
- **Amministrazione** dei dati
- Riduzione di **tempi** e **costi** di sviluppo

Al contrario però sono **complessi** e costosi da gestire perché rendono necessaria la definizione di uno **schema** e possono solo gestire dati **strutturati** ed **omogenei**.

# 2 Modelli di dati

Progettare una BD significa progettare la struttura dei dati e le applicazioni. Per farlo al meglio è fondamentale rappresentare in modo **astratto** e simbolico il dominio del discorso tramite la **modellazione**.

**Definizione 2.0.1** (Modello astratto)**.** *Un modello astratto è la* ***rappresentazione formale*** *di idee e conoscenze relative ad un fenomeno tramite un* ***linguaggio formale*** *a seguito di una* ***interpretazione*** *soggettiva.*

## 2.1 Ruoli

I ruoli principali nella modellazione sono:

- **Committente**: persona con l'esigenza
- **Progettista** o **analista**: crea un progetto concettuale
- **Programmatore**: sviluppano la BD e le applicazioni
- **DB Administrator**: gestisce gli utenti e il sistema

## 2.2 Fasi

Le fasi della progettazione sono:

1. **Analisi dei requisiti**: definizione dei bisogni del committente
2. **Progettazione concettuale**: traduzione dei requisiti in un progetto, struttura concettuale dei dati, dei vincoli e delle operazioni
3. **Progettazione logica**: traduzione dello *schema concettuale* nello schema logico, che è espresso nel modello dei dati del sistema scelto
4. **Progettazione fisica**: produce lo schema fisico che arricchisce quello logico con specifiche sull'organizzazione fisica dei dati


## 2.3 Aspetto ontologico

Questo aspetto si concentra su quale conoscenza del dominio deve essere rappresentata. In particolare la conoscenza può essere **astratta**, **concreta** o **procedurale** (operazioni di base e comunicazione).

### 2.3.1 Conoscenza concreta

La conoscenza concreta riguarda i **fatti** specifici che si vogliono rappresentare: **entità**, **collezioni** e **associazioni**.

**2.3.1.1 Entità e proprietà** Le **entità** sono ciò di cui ci interessa rappresentare alcuni fatti o proprietà (e.g. un libro) mentre le **proprietà** sono fatti che descrivono caratteristiche di determinate entità (e.g. titolo).

*Note* 2.3.1. Un'entità non coincide con l'insieme dei valori assunti dalle sue proprietà, in quanto queste possono cambiare nel tempo o possono essere identiche pur essendo due entità diverse. E.g. una persona la cui età aumenta ogni anno o due persone con stesso nome, età e indirizzo.

Una **proprietà** è una coppia *nome* e *valore*, dove quest'ultimo è di un certo tipo e all'interno di un **dominio** di possibili valori. Si possono classificare come:

- **Atomica** se il suo valore non è scomponibile, altrimenti **strutturata**
- **Univoca** se il suo valore è unico, altrimenti **multivalore**
- **Totale** o **obbligatoria** se ogni entità nell'universo assume un valore per essa, altrimenti **parziale** o **opzionale**
- **Costante** o **variabile**
- **Calcolata** o **non calcolata**

Un **tipo di entità** è una descrizione astratta di ciò che accomuna un insieme di entità omogenee, esistenti o possibili. È quindi un insieme infinito. E.g. persona, auto, esame.

**2.3.1.2 Collezione** Una collezione è un insieme **variabile** nel tempo di entità omogenee interessanti nel dominio. L'insieme degli elementi di una collezione in un dato momento è detto **estensione** della collezione. A differenza del *tipo di entità*, questi insiemi sono finiti.

**2.3.1.3 Associazione**

**Definizione 2.3.1** (Istanza di associazione)**.** *Un'**istanza di associazione** è un fatto che correla due o più entità stabilendo un legame logico tra di loro.*

**Definizione 2.3.2** (Associazione)**.** *L'**associazione** $R(X, Y)$ fra due collezioni di entità $X$ e $Y$ è quindi l'insieme di istanze di associazione tra gli elementi di $X$ e $Y$ che varia nel tempo.*

**Definizione 2.3.3** (Prodotto cartesiano)**.** *Il **prodotto cartesiano** $(X \times Y)$ è il dominio dell'associazione.*

**Osservazione 2.3.1.** Se vediamo due collezioni $X$ e $Y$ come due insiemi, un'istanza di associazione tra di loro può essere vista come una coppia di elementi $(x; y)$, con $x \in X$ e $y \in Y$, e quindi un'associazione $R$ tra $X$ e $Y$ può essere vista come un sottoinsieme del prodotto $X \times Y$, ovvero come una relazione matematica tra tali insiemi.

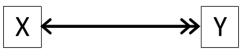
Le associazioni hanno due caratteristiche strutturali:

- **Molteplicità**

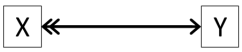
  **Definizione 2.3.4** (Vincolo di univocità)**.** *Un'associazione $R(X, Y)$ è **univoca** rispetto ad $X$ se per ogni elemento $x \in X$ esiste al più un elemento di $Y$ che è associato ad $x$. Altrimenti è **multivalore** rispetto ad $X$.*

  Questo ci porta alla **cardinalità** di un'associazione, che può essere:

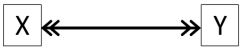
  - **Uno a molti**: $R(X, Y)$ è $(1 : N)$ se essa è multivalore su $X$ ed univoca su $Y$



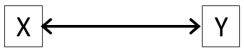
– **Molti a uno**: $R(X,Y)$ è $(N:1)$ se essa è univoca su $X$ e multivalore




– **Molti a molti**: $R(X,Y)$ è $(N:M)$ se essa è multivalore su $X$ e $Y$

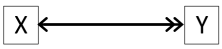



– **Uno a uno**: $R(X,Y)$ è $(1:1)$ se essa è univoca su $X$ e $Y$

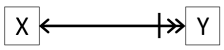



- **Totalità**

  **Definizione 2.3.5** (Vincolo di totalità)**.** *Un'associazione $R(X,Y)$ è **totale** (o surgettiva) su $X$ se per ogni elemento $x \in X$ esiste almeno un elemento di $Y$ che è associato ad $x$. Altrimenti è **parziale** rispetto ad $X$.*




Totale




Parziale

Un'associazione si rappresenta come visto e con l'aggiunta di un'**etichetta** con il suo nome, scelto di solito con un *predicato*.

### 2.3.2 Conoscenza astratta

La conoscenza astratta riguarda i fatti generali che descrivono la **struttura** della conoscenza concreta, le **restrizioni** sui valori possibili della conoscenza concreta e sui **vincoli d'integrità** e le **regole** per dedurre nuovi fatti.

**Definizione 2.3.6** (Modello dei dati)**.** *Un modello dei dati è un insieme di meccanismi di astrazione per descrivere la struttura della conoscenza concreta.*

**Definizione 2.3.7** (Schema)**.** *Uno schema è la descrizione della struttura della conoscenza concreta e dei vincoli di integrità usando un particolare modello di dati.*

*Note* 2.3.2. Come notazione grafica per lo schema usiamo una **variante** del modello ER.

**2.3.2.1 Oggetti** Ad ogni **entità** del dominio corrisponde un oggetto che può rispondere a dei **messaggi** (anche chiamati **attributi**), restituendo valori memorizzati o calcolati tramite procedure.

**Definizione 2.3.8** (Oggetto)**.** *Un oggetto è un'entità software che modella un'entità dell'universo e che ha:*

- ***Stato**: modellato da un insieme di costanti o variabili con valori di qualsiasi complessità*
- ***Comportamento**: un insieme di procedure locali chiamate **metodi**, che modellano le operazioni di base che riguardano l'oggetto e le proprietà derivabili da altre*
- ***Identità***

Il **tipo oggetto** definisce l'insieme dei messaggi a cui può rispondere un insieme di possibili oggetti. Tra i tipi oggetto può essere definita una **relazione di sottotipo** che ha le seguenti proprietà:

- È **asimmetrica**, **riflessiva** e **transitiva** (relazione di *ordine parziale*)
- **Sostitutività**: se $T$ è sottotipo di $T'$ allora gli elementi di $T$ possono essere usati in ogni contesto i cui possano apparire quelli di $T'$. In particolare:
  – Gli elementi di $T$ hanno tutte le **proprietà** di quelli di $T'$
  – Per ogni **proprietà** $p \in T'$, il suo tipo $T$ è un sottotipo del suo tipo in $T'$

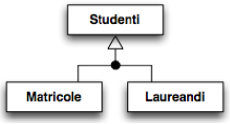
**2.3.2.2 Classe** Una classe è un insieme di oggetti dello stesso tipo, modificabile con operatori per includere o estrarre elementi dall'insieme.
Spesso le classi sono organizzate in una **gerarchia** di **specializzazione** o **generalizzazione** (**sottoclassi** e **superclassi**). Queste hanno due caratteristiche:

- **Ereditarietà** delle proprietà che permette di definire:
  - un tipo oggetto a partire da un altro
  - l'implementazione di un tipo oggetto a partire da un'altra implementazione. In questo caso gli attributi possono solo essere **aggiunti** o **ridefiniti** solo specializzandone il tipo
- Gli elementi di una sottoclasse sono un **sottoinsieme** di quelli della superclasse. Questa relazione ha le seguenti proprietà:
  - È **asimmetrica**, **riflessiva** e **transitiva**
  - **Vincolo intensionale**: se $C$ è sottoclasse di $C'$ allora il tipo degli elementi di $C$ è sottotipo degli elementi di $C'$
  - **Vincolo estensionale**: se $C$ è sottoclasse di $C'$ allora gli elementi di $C$ sono un sottoinsieme degli elementi di $C'$

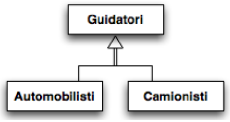
I vincoli sugli insieme di sottoclassi possono essere di **disgiunzione** e di **copertura**. Questo porta ad avere quattro tipi di relazioni tra sottoinsiemi:

- **Scorrelate**: non richiedono nessun vincolo e possono essere rappresentate in due modi

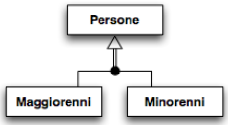

- **Disgiunte**




- **Copertura**

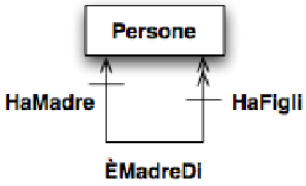



- **Partizione**



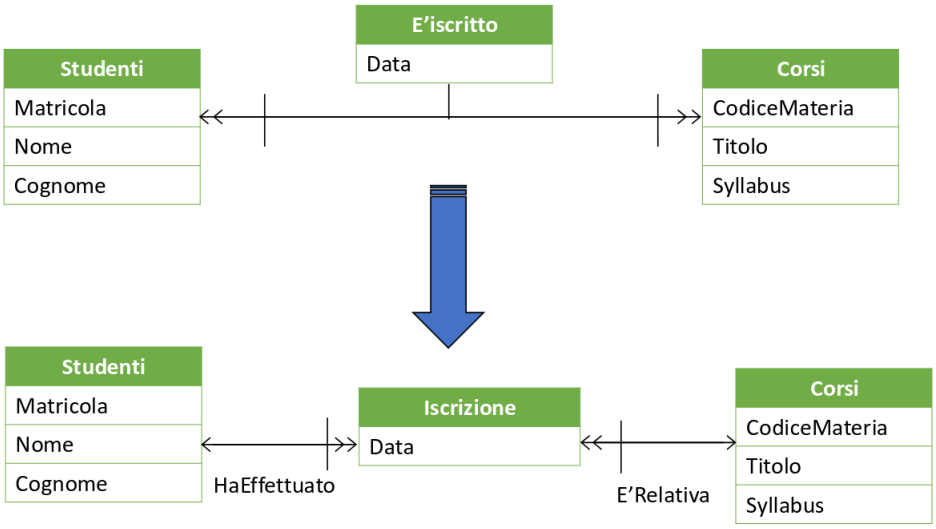

È possibile avere l'**ereditarietà multipla** definendo un tipo per ereditarietà da più supertipi. Bisogna prestare attenzione quando un attributo viene ereditato da più antenati.

**2.3.2.3 Associazioni** Le associazioni si modellano con un costrutto apposito e possono avere delle **proprietà** ed essere **ricorsive**. L'ultimo caso si presenta quando abbiamo relazioni binarie fra gli elementi di una stessa collezione. In questo caso bisogna etichettare anche i ruoli agli estremi della freccia.

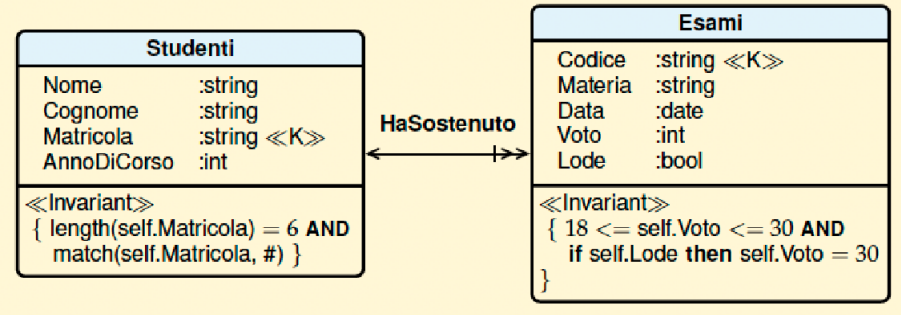



*Note* 2.3.3. È possibile avere più associazioni tra classi diverse che rappresentano informazioni diverse.

**Osservazione 2.3.2** (Reificazione)**.** È possibile trasformare un'associazione tra due classi in una situazione con tre classi e due associazioni. Ad esempio:




**2.3.2.4 Restrizioni** I **vincoli d'integrità** impongono restrizioni sui possibili valori della conoscenza concreta. Possono essere **statici** o **dinamici** e arricchiscono la descrizione di una classe.



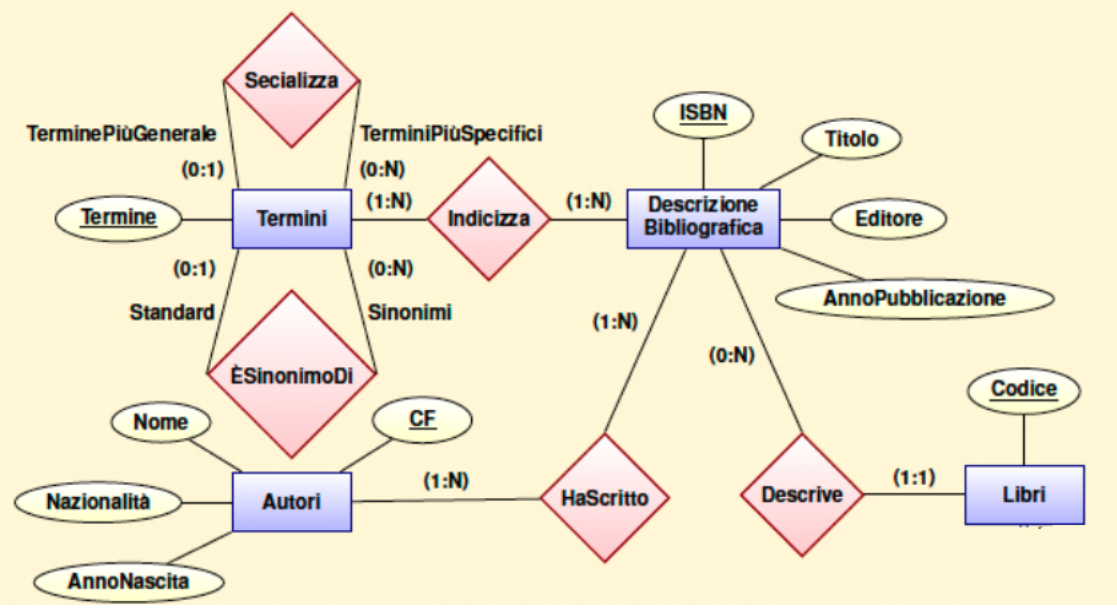

*Note* 2.3.4. Gli attributi marcati con $<< K >>$ sono una **chiave**.

## 2.4 Modello entità-relazione

È Il modello più popolare per il disegno concettuale di BD. Non tratta gerarchie di inclusione tra collezioni, non distingue collezioni e tipi e non supporta alcun meccanismo di ereditarietà. Definisce un meccanismo per modellare direttamente le associazioni non binarie o con proprietà.
Prevede due meccanismi di **astrazione**:

- Modellare **insiemi di entità**, con le relative proprietà
- Modellare le associazioni (chiamate **relazioni**).

Le collezioni sono chiamate **tipi di entità**, e gli attributi dei loro elementi possono assumere solo valori di tipo primitivo.




## 2.5 Modello relazionale

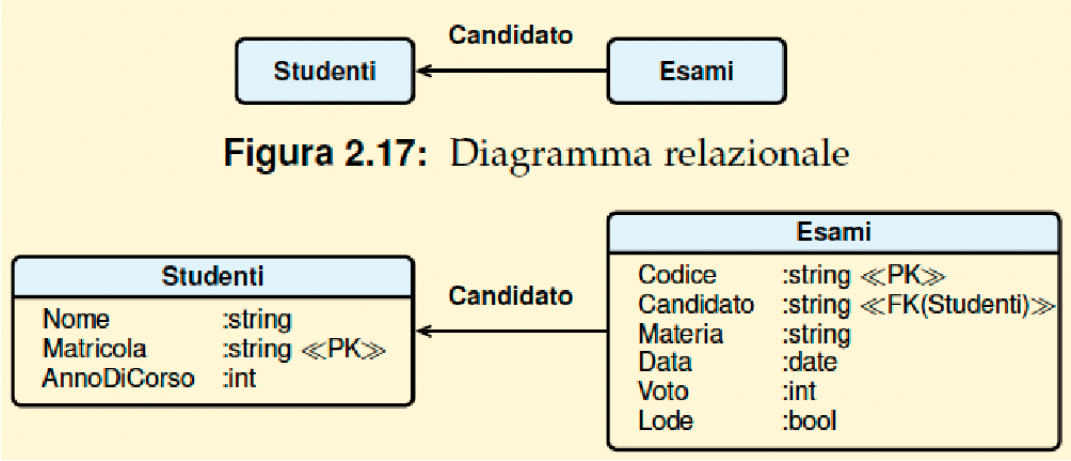
E' il modello dei dati usato dagli attuali sistemi commerciali. I meccanismi per definire una base di dati con questo modello sono l'**ennupla** e la **relazione**.

**Definizione 2.5.1** (Ennupla)**.** *Un tipo ennupla è un insieme di coppie (attributo, tipo primitivo) ed un valore di tipo ennupla è un insieme di coppie (attributo, valore), dette anche campi, con gli stessi attributi del tipo e in cui il valore di ogni attributo appartiene al corrispondente tipo primitive.*

**Definizione 2.5.2** (Relazione)**.** *Una relazione è un insieme di ennuple con lo stesso tipo.*

**Definizione 2.5.3** (Superchiave e chiave)**.** *Un insieme di attributi i cui valori determinano univocamente un'ennupla di una relazione $R$ è una **superchiave** per $R$. Una superchiave tale che togliendo un qualunque attributo essa non sia più una superchiave è una **chiave** per $R$. Tra le chiavi di $R$ ne viene scelta una come chiave **primaria**.*

Le **associazioni** tra i dati sono rappresentate attraverso opportuni attributi, chiamati **chiavi esterne**, che assumono come valori quelli della chiave primaria di un'altra relazione.



# 3 Progettazione

Progettare una BD significa definire lo **schema globale** dei dati, i **vincoli di integrità** e le **operazioni** delle applicazioni allo scopo di prepararsi alla realizzazione. Si articola in tre fasi:

1. **Analisi dei requisiti**:
   - Analizza il sistema esistente e raccoglie **requisiti informali**
   - Elimina **ambiguità**, **imprecisioni** e **disuniformità** cercando sinonimi e omonimi e unificandoli
   - Raggruppa le frasi relative a diverse categorie di **dati**, **vincoli**, e **operazioni**
   - Definisce un **glossario**
   - Disegna lo **schema di settore**
   - Specifica le **operazioni** e ne verifica la **coerenza** con i dati
2. **Progettazione**:
   - Concettuale, logica e fisica dei dati. Identificare **classi** (e gli attributi e tipi), associazioni (e le loro proprietà), elencare le **chiavi**, individuare le **sottoclassi** e le **generalizzazioni**
   - Delle applicazioni
3. **Realizzazione**

## 3.1 Documentazione

Dato che il linguaggio naturale è pieno di **ambiguità** è fondamentale evitarle. Questo è possibile con l'aiuto delle seguenti regole per scrivere una buona **documentazione**:

- Studiare e comprendere il sistema informativo e i bisogni di tutti i settori dell'organizzazione
- Scegliere il corretto **livello di astrazione**
- **Standardizzare** la struttura delle frasi
- **Suddividere** le frasi articolate
- **Separare** le frasi sui dati da quelle sulle funzioni

**Esempio 3.1.1** (Società di formazione)**.** Si vuole realizzare una base di dati per una società che eroga corsi, di cui vogliamo rappresentare i dati dei partecipanti ai corsi e dei docenti. Per gli studenti (circa 5000), identificati da un codice, si vuole memorizzare il codice fiscale, il cognome, l'età, il sesso, il luogo di nascita, il nome dei loro attuali datori di lavoro, i posti dove hanno lavorato in precedenza insieme al periodo, l'indirizzo e il numero di telefono, i corsi che hanno frequentato (i corsi sono in tutto circa 200) e il giudizio finale.
Rappresentiamo anche i seminari che stanno attualmente frequentando e, per ogni giorno, i luoghi e le ore dove sono tenute le lezioni. I corsi hanno un codice, un titolo e possono avere varie edizioni con date di inizio e fine e numero di partecipanti. Se gli studenti sono liberi professionisti, vogliamo conoscere l'area di interesse e, se lo possiedono, il titolo. Per quelli che lavorano alle dipendenze di altri, vogliamo conoscere invece il loro livello e la posizione ricoperta.
Per gli insegnanti (circa 300), rappresentiamo il cognome, l'età, il posto dove sono nati, il nome del corso che insegnano, quelli che hanno insegnato nel passato e quelli che possono insegnare. Rappresentiamo anche tutti i loro recapiti telefonici. I docenti possono essere dipendenti interni della società o collaboratori esterni.

Per prima cosa abbiamo evidenziato dello stesso colore i **sinonimi** e le parole utilizzate in più **contesti** diversi in modo da poterli unificare e costruire il glossario.

| Termine | Descrizione | Sinonimi | Collegamenti |
|---|---|---|---|
| Partecipante | Persona che partecipa ai corsi | Studente | Corso, società |
| Docente | Docente dei corsi. Può essere esterno | Insegnante | Corso |
| Corso | Corso organizzato dalla società. Può avere più edizioni. | Seminario | Docente |
| Datori | Ente presso cui i partecipanti lavorano o hanno lavorato. | Posti | Partecipante |

Procediamo poi trovando gli **attributi** per ogni classe.

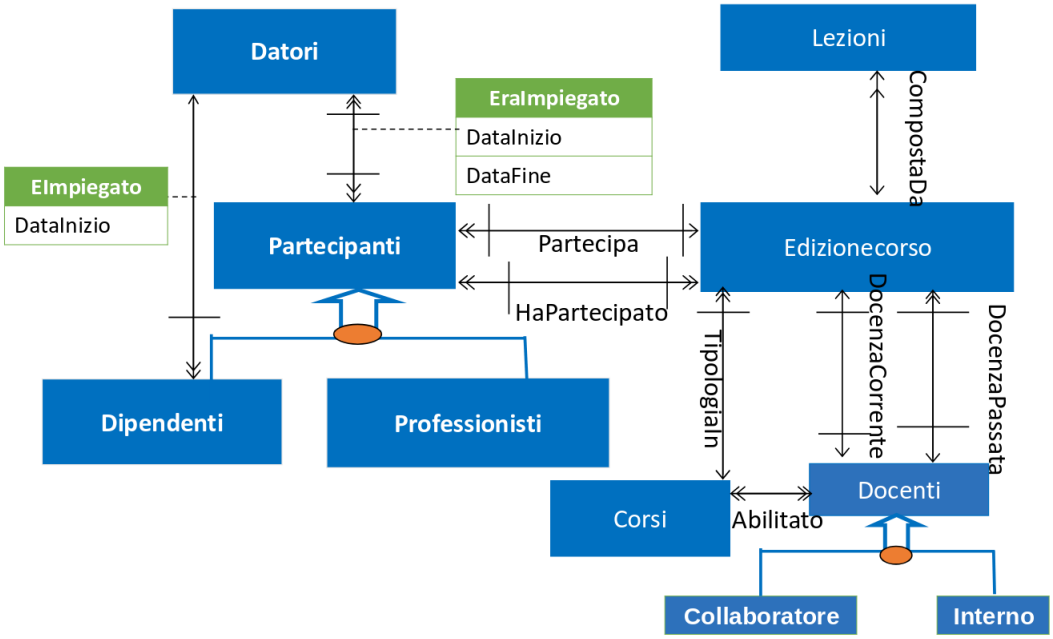
| **Partecipanti** |
|---|
| Codice |
| CodiceFiscale |
| Cognome |
| Nome |
| Genere |
| CittaNascita |
| DataNascita |

| **Docenti** |
|---|
| CodiceFiscale |
| Cognome |
| Nome |
| Genere |
| CittaNascita |
| DataNascita |
| Recapiti |

| **Corsi** |
|---|
| Codice |
| Titolo |

| **DatoriLavoro** |
|---|
| CodiceFiscale |
| Cognome |
| Nome |
| Indirizzo |
| Citta |
| Telefono |

Infine definiamo le **relazioni** tra le classi.



# 4 Modello relazionale

Il modello relazionale fu presentato da E. F. Codd nel 1970 per favorire l'**indipendenza** dei dati. Si basa sul concetto di **relazione**, la quale ha come rappresentazione la **tabella**.
In ogni base di dati distinguiamo lo **schema relazionale**, invariato nel tempo e che descrive la struttura) e l'**istanza**, ovvero i valori attuali.

## 4.1 Matematica

In matematica definiamo una **relazione** come l'insieme dei **domini**

$$D_1, \ldots, D_n$$

Inoltre il **prodotto cartesiano** $D_1 \times \ldots \times D_n$è l'insieme di tutte le **n-uple** $(d_1, \ldots, d_n)$ tali che

$$d_1 \in D_1, \ldots, d_n \in D_n$$

**Esempio 4.1.1** (Relazioni in matematica)**.** Dati i domini

$$D_1 = \{a, b\} \qquad D_2 = \{x, y, z\}$$

il prodotto cartesiano è

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

mentre una relazione possibile è

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

## 4.2 Modello

I meccanismi per definire una BD con il modello relazionale sono la **ennupla** (insieme finito di coppie (Attributo, Tipo elementare)) e la **relazione**.

**Definizione 4.2.1** (Schema di relazione)**.** *Uno schema di relazione $R(T)$ è una coppia formata da un **nome** $R$ e da un **tipo** $T$ definito come segue:*

- *int, real, boolean e string sono tipi **primitivi***
- *$T = (A_1 : T_1, \ldots, A_n : T_n)$ è un tipo **ennupla** di **grado** $n$ se $T_1, \ldots, T_n$ sono tutti tipi primitivi e se $A_1, \ldots, A_n$ sono etichette distinte dette **attributi***
- *Due ennuple sono uguali se hanno uguale il grado, gli attributi e il tipo degli attributi con lo stesso nome*
- *L'ordine degli attributi non importa*
- *Se $T$ è tipo ennupla allora $\{T\}$ è un insieme di ennuple o tipo **relazione***
- *Due tipi relazione sono uguali se hanno lo stesso tipo ennupla*

**Definizione 4.2.2** (Schema relazionale)**.** *Uno schema relazionale è costituito da schemi di relazione $R_i : \{T_i\} \quad i = 1, \ldots, k$ e da un insieme di relativi **vincoli di integrità**.*

**Definizione 4.2.3** (Ennupla)**.** *Un'ennupla $t = (A_1 : V_1, \ldots, A_n : V_n)$ di tipo $T = (A_1 : T_1, \ldots, A_n : T_n)$ è un insieme di coppie $(A_i, V_i)$ con $V_i$ di tipo $T_i$. Due ennuple sono uguali se hanno lo stesso insieme di coppie.*

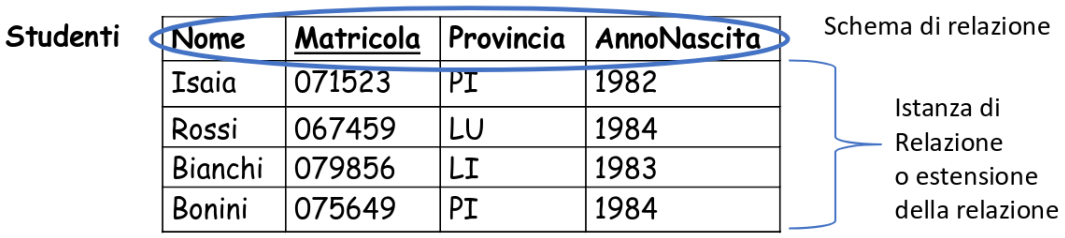
**Definizione 4.2.4** (Istanza)**.** *Un'istanza dello schema $R_i : \{T_i\}$ o **relazione** è un insieme finito di ennuple $\{t_1, t_2, \ldots, t_k\}$, con ti di tipo $T_i$. La sua **cardinalità** è il numero delle sue ennuple. L'istanza di uno schema relazionale è formata da un'istanza di ogni suo schema di relazione.*

### 4.2.1 Tabella

Una tabella rappresenta una **relazione** se:

- I valori di ogni colonna sono fra loro omogenei
- Le righe sono diverse fra loro
- Le intestazioni delle colonne sono diverse tra loro

L'ordinamento di righe e colonne non è importante.




## 4.3 Valori

Il modello relazionale è basato su **valori**, ovvero i riferimenti fra i dati in relazioni diverse sono rappresentati per mezzo di valori dei domini che compaiono nelle ennuple. Questo permette di mantenere **indipendenza** dalle strutture fisiche, si rappresenta solo i dati necessari e si garantisce **portabilità**.

**Definizione 4.3.1** (Valore nullo)**.** *Il valore nullo denota l'assenza di un valore del dominio.*

**Definizione 4.3.2** (Valore)**.** $t[A]$*, per ogni attributo* $A$*, è un valore nel dominio* $dom(A)$ *oppure il valore nullo* $NULL$*.*

## 4.4 Vincoli di integrità

Uno schema relazionale è costituito da un insieme di **schemi di relazione** e da un insieme di **vincoli d'integrità** su i possibili valori delle estensioni delle relazioni. Questi ultimi permettono di descrivere più accuratamente la realtà migliorando la **qualità dei dati** e aiutando nella progettazione

**Definizione 4.4.1** (Vincolo d'integrità)**.** *Un vincolo d'integrità è una proprietà che deve essere soddisfatta dalle istanze che rappresentano informazioni corrette per l'applicazione. È espresso mediante una funzione booleana che associa ad ogni istanza il valore vero o falso.*

Esistono due tipi di vincoli:

- **Intrarelazionali**: devono essere rispettati dai valori contenuti nella relazione considerata. Possono essere sui valori o sulle ennuple.
- **Interrelazionali**: devono essere rispettati da valori contenuti in relazioni diverse

### 4.4.1 Vincoli di ennupla

Questi vincoli esprimono condizioni sui valori di ciascuna ennupla, indipendentemente dalle altre. Quando coinvolgono un solo attributo sono chiamati **vincoli di dominio**.

## 4.5 Chiave

Una chiave è un insieme di attributi che identificano le ennuple di una relazione.

**Definizione 4.5.1** (Chiave)**.** *Un insieme $K$ di attributi per uno schema di relazione $r$ è:*

- ***Superchiave*** *se $r$ non contiene due ennuple distinte $t_1$ e $t_2$ con $t_1[K] = t_2[K]$*
- ***Chiave*** *se è una superchiave* ***minimale****, cioè se non contiene un'altra superchiave*

**Definizione 4.5.2** (Chiave primaria)**.** *La chiave primaria di uno schema di relazione è una delle chiavi, di solito quella con il minor numero di attributi. Non ammette valori nulli ed è indicata con la sottolineatura.*

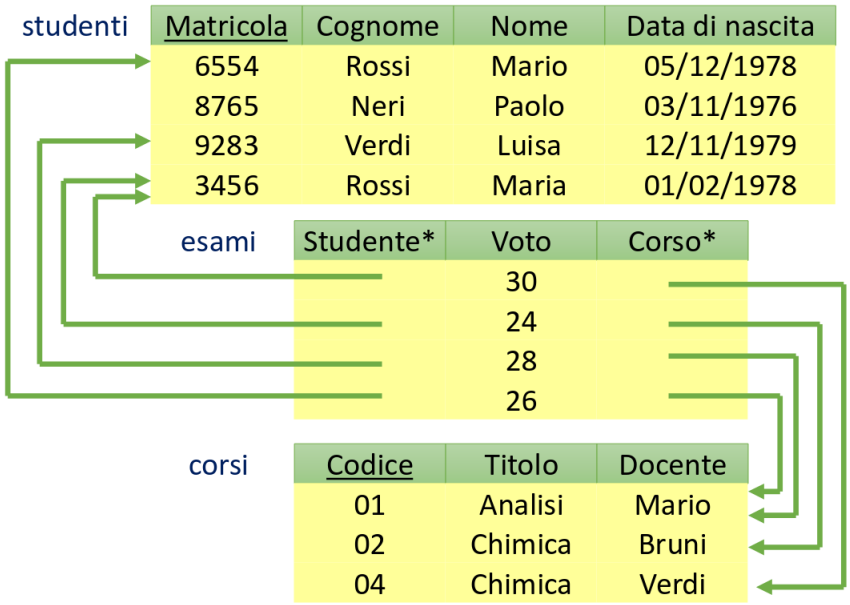
| Matricola | Cognome | Nome | Corso | Nascita |
|---|---|---|---|---|
| 86765 | NULL | Mario | Ing Inf | 5/12/78 |
| 78763 | Rossi | Mario | Ing Civile | 3/11/76 |
| 65432 | Neri | Piero | Ing Mecc | 10/7/79 |
| 87654 | Neri | Mario | Ing Inf | NULL |
| 43289 | Neri | Mario | NULL | 5/12/78 |

*Note* 4.5.1. È possibile che esistano degli insiemi di attributi che soddisfino casualmente tutti i vincoli per essere chiavi, ma questo deve succedere **sempre** per tutte le istanze.

**Osservazione 4.5.1** (Esistenza)**.** Ogni relazione ha come superchiave l'insieme di tutti gli attributi su cui è definita, quindi ha sempre almeno una chiave.

## 4.6 Integrità referenziale

Nel modello relazionale le informazioni in relazioni sono correlate attraverso **valori** comuni, spesso quelli delle **chiavi primarie**. Deve quindi esserci una **coerenza**.
Gli attributi che permettono le correlazioni sono indicati sia con la sottolineatura (**chiavi esterne**) che con l'asterisco.



### 4.6.1 Integrità referenziale

**Definizione 4.6.1** (Vincolo di integrità referenziale)**.** *Un vincolo di integrità referenziale (**foreign key**) fra gli attributi $X$ di una relazione $R_1$ e un'altra relazione $R_2$ impone ai valori su $X$ in $R_1$ di comparire come valori della chiave primaria di $R_2$.*

In caso di violazione del vincolo di integrità (e.g. viene eliminata una ennupla dalla tabella riferita), è possibile:

- Rifiutare l'operazione
- Eliminare in cascata nelle altre tabelle
- Introdurre valori nulli

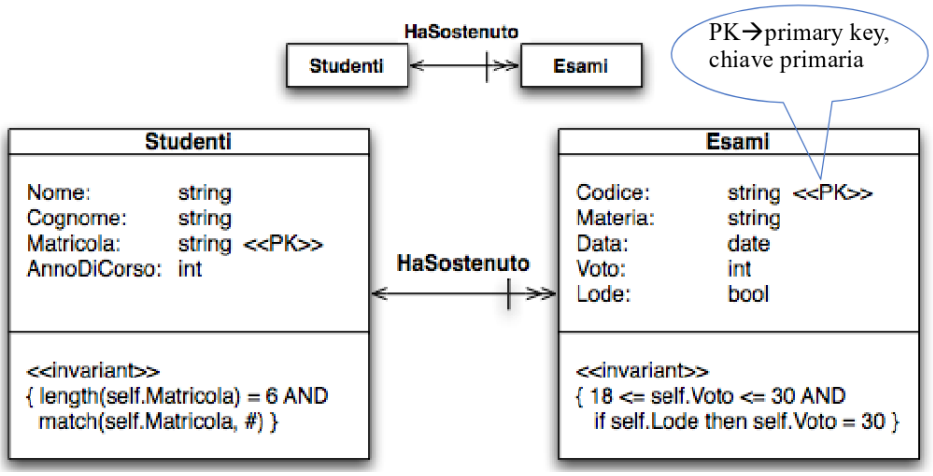
*Note* 4.6.1. È possibile avere più chiavi esterne e in questo caso ci sono **vincoli multipli**.

# 5 Trasformazione di schemi

L'obiettivo della trasformazione è quello di ottenere da uno schema concettuale uno schema **logico-relazionale** che rappresenti gli stessi dati in maniera **corretta** ed **efficiente**, riducendo la ridondanza e facilitandone la comprensione.
Questa trasformazione prende in **ingresso** lo schema relazionale, il carico applicativo e il modello logico e restituisce in **uscita** uno schema logico e la documentazione.
Si compone dei seguenti passi:

1. Rappresentazione delle associazioni uno ad uno e uno a molti
2. Rappresentazione delle associazioni molti a molti o non binarie
3. Rappresentazione delle gerarchie di inclusione
4. Identificazione delle chiavi primarie
5. Rappresentazione degli attributi multivalore
6. Appiattimento degli attributi composti

**Esempio 5.0.1** (Esempio di trasformazione)**.** Dato il seguente schema concettuale




per ottenere uno schema logico introduciamo l'attributo **matricola**. Questo ci porta ad avere due relazioni collegate dal nuovo attributo.

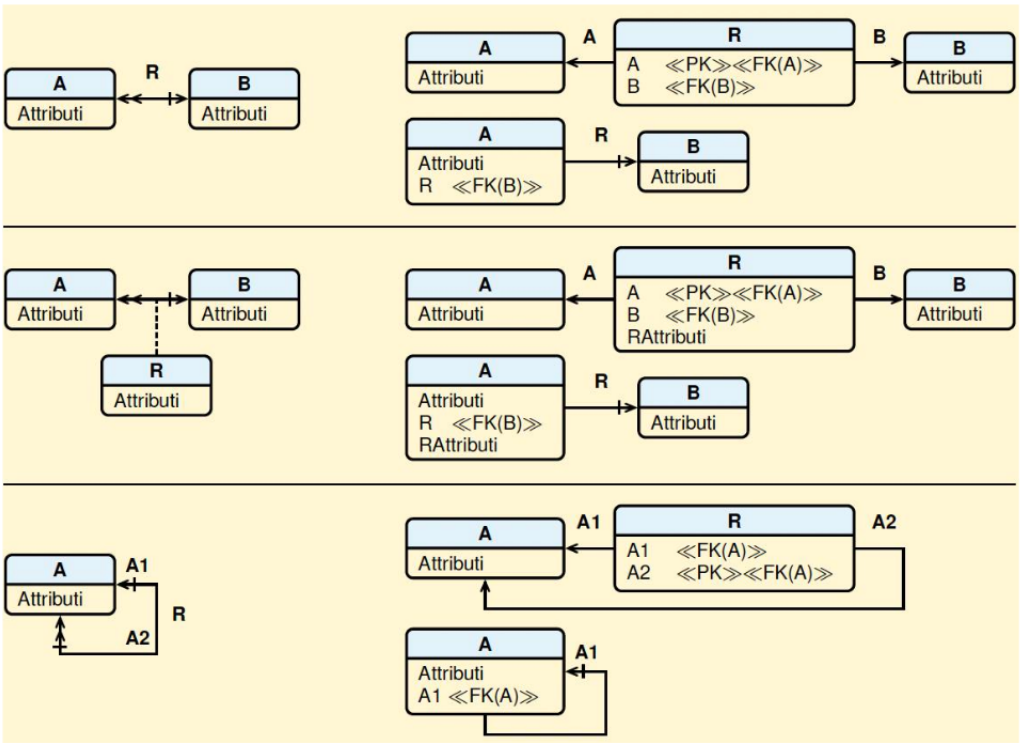
| **Nome** | **Matricola** | **Provincia** | **AnnoNascita** |
|---|---|---|---|
| Isaia | 071523 | PI | 1982 |
| Rossi | 067459 | LU | 1984 |
| Bianchi | 079856 | LI | 1983 |
| Bonini | 075649 | PI | 1984 |

| **Materia** | **Candidato*** | **Data** | **Voto** |
|---|---|---|---|
| BD | 071523 | 12/01/06 | 28 |
| BD | 067459 | 15/09/06 | 30 |
| FP | 079856 | 25/10/06 | 30 |
| BD | 075649 | 27/06/06 | 25 |
| LMM | 071523 | 10/10/06 | 18 |

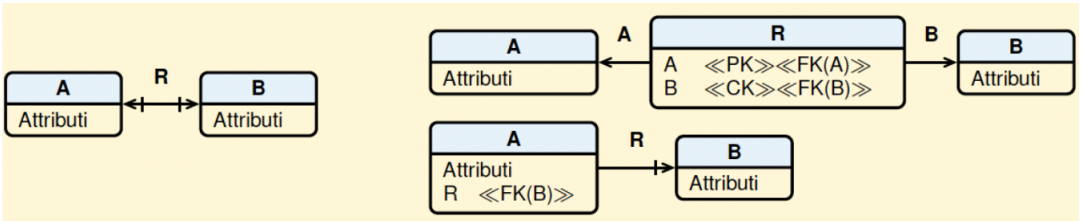
## 5.1 Rappresentazioni

### 5.1.1 Uno a molti/uno

**5.1.1.1 Uno a molti** Le associazioni uno a molti si rappresentano aggiungendo agli attributi della relazione rispetto a cui l'associazione è univoca una **chiave esterna** che riferisce l'altra relazione. Se l'associazione ha degli attributi, questi si aggiungono alla relazione in cui è presente la chiave esterna.




**5.1.1.2 Uno ad uno** Le associazioni uno a uno si rappresentano aggiungendo la **chiave esterna** ad una qualunque delle due relazioni che riferisce l'altra relazione, preferendo quella rispetto a cui l'associazione è totale, nel caso in cui esista un vincolo di totalità.




**5.1.1.3 Vincoli**

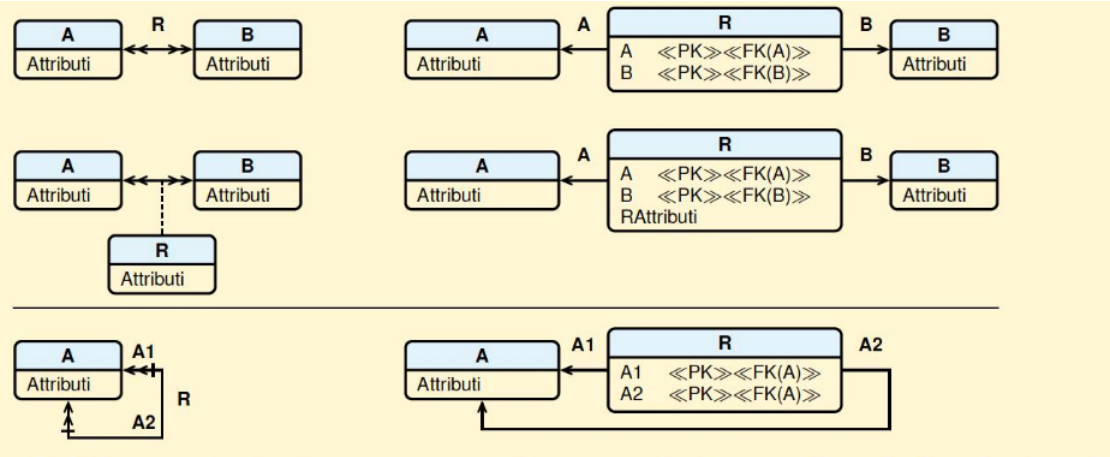
**Definizione 5.1.1** (Diretta)**.** *La direzione dell'associazione rappresentata dalla chiave esterna è detta* ***la diretta*** *dell'associazione.*

I vincoli sulla **cardinalità** delle associazioni **uno ad uno** e **uno a molti** sono:

- **Univocità** della diretta
- **Totalità** della diretta: vincolo *not null* sulla chiave esterna
- **Univocità** dell'inversa e **totalità** della diretta: vincolo *not null* e *di chiave* sulla chiave esterna

### 5.1.2 Molti a molti

Un'associazione molti a molti si rappresenta aggiungendo allo schema una nuova relazione che contiene due chiavi esterne che riferiscono le due relazioni coinvolte; la chiave primaria di questa relazione è costituita dall'insieme di tutti i suoi attributi.




### 5.1.3 Gerarchie tra classi

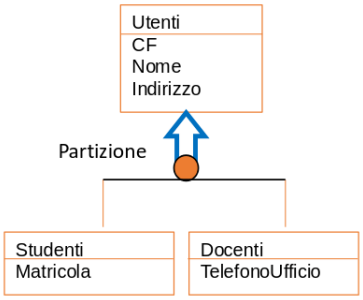
Il modello relazionale non può rappresentare le gerarchia tra classi. Bisogna quindi eliminarle e sostituirle con classi e relazioni usando le seguenti tecniche:

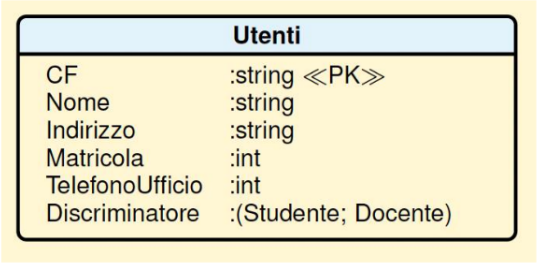
- **Relazione unica**: accorpamento delle figlie nel genitore
- **Partizionamento orizzontale**: accorpamento del genitore nelle figlie
- **Partizionamento verticale**: sostituzione della gerarchia con relazioni

#### 5.1.3.1 Relazione unica

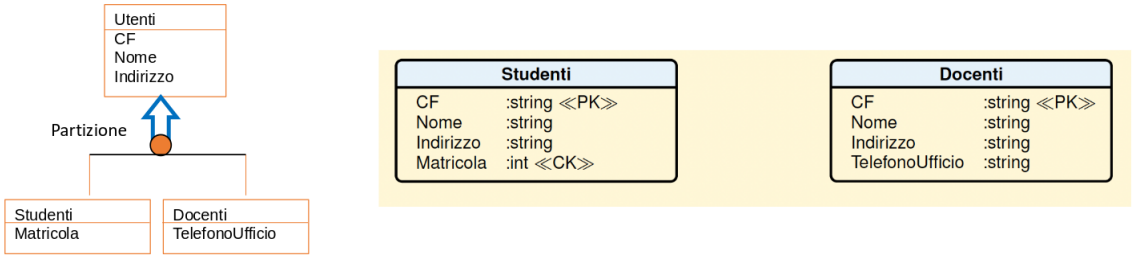
Si seguono i seguenti passi:

1. Se $A_0$ è la classe genitore di $A_1$ ed $A_2$, queste ultime vengono eliminate ed accorpate alla prima
2. Ad $A_0$ viene aggiunto un attributo (**discriminatore**) che indica da quale delle classi figlie deriva una certa istanza, e gli attributi di $A_1$ ed $A_2$ vengono assorbiti dalla classe genitore, e assumono valore nullo sulle istanze provenienti dall'altra classe
3. Infine, una relazione relativa a solo una delle classi figlie viene acquisita dalla classe genitore e avrà comunque cardinalità minima uguale a 0, in quanto gli elementi dell'altra classe non contribuiscono alla relazione.



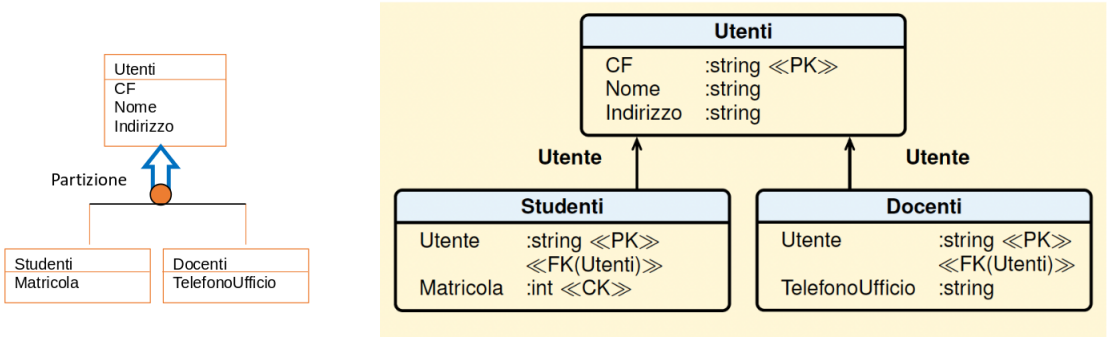



**5.1.3.2 Partizionamento orizzontale** Se $A_0$ è la classe genitore di $A_1$ ed $A_2$, la classe genitore viene eliminata, e le classi figlie ereditano le proprietà (attributi,identificatore e relazioni) della classe genitore.




*Note* 5.1.1. Questa tecnica divide gli elementi della superclasse $A_0$ in più relazioni diverse, per cui non è possibile mantenere un vincolo referenziale verso $A_0$. In conclusione, questa tecnica non si usa se nello schema relazionale c'è una associazione diretta verso $A_0$, ovvero che entra nella superclasse.

**5.1.3.3 Partizionamento verticale** In questo caso non c'è un trasferimento di attributi o di associazioni e le classi figlie $A_1$ ed $A_2$ sono identificate esternamente dalla classe genitore $A_0$. Nello schema ottenuto vanno aggiunti dei vincoli: ogni occorrenza di $A_0$ non può partecipare contemporaneamente alle due associazioni, e se la gerarchia è totale, deve partecipare ad almeno una delle due.




### 5.1.4 Chiavi primarie

È necessario definire per ogni relazione un insieme di attributi che funga da chiave primaria, seguendo questi passi:

1. Si considerano le relazioni che corrispondono a classi dello schema originale che erano prive di superclassi (**classi radice**). La chiave primaria è di norma un attributo artificiale, tipicamente un numero progressivo assegnato dal sistema. E' possibile utilizzare un attributo presente nella classe, purché l'attributo sia **univoco**, **totale** e **costante**.

2. Per ogni relazione dello schema che corrisponde ad una **sottoclasse** dello schema originario, la chiave primaria sarà la stessa della superclasse.

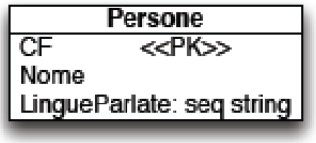
3. Per le relazioni che corrispondono ad $\mathbf{N:M}$ nello schema originario, la chiave primaria sarà costituita dalla concatenazione delle chiavi esterne.

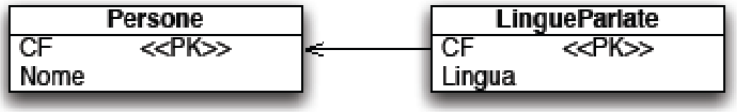
### 5.1.5 Attributi multivalore

Una proprietà multivalore di una classe $C$ si rappresenta eliminando il corrispondente attributo da $C$ e creando una nuova relazione $N$ con una chiave di due attributi:

- una **chiave esterna** che fa riferimento alla chiave primaria di $C$
- un **attributo** che corrisponde all'attributo multivalore da trasformare

Un oggetto di $C$ con chiave primaria $k$ ed in cui l'attributo assume valore $a_1, \ldots, a_n$ si rappresenta poi inserendo nella relazione $N$ le coppie $(k, a_1), \ldots, (k, a_n)$



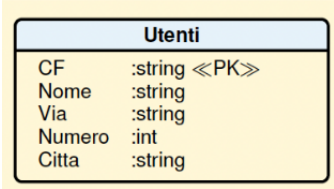

### 5.1.6 Attributi composti

Se un attributo $A_i$ di uno schema di relazione è di tipo $[A_{i1} : T_{i1}, \ldots, A_{ij} : T_{ij}]$, si sostituisce con gli attributi $A_{i1} : T_{i1}, \ldots, A_{ij} : T_{ij}$. Se $A_i$ faceva parte della chiave primaria dello schema di relazione, si sostituisce $A_i$ con gli attributi $A_{i1}, \ldots, A_{ij}$ nella chiave, e poi si verifica che non esista un sottoinsieme degli attributi della nuova chiave primaria che è esso stesso una chiave.

**Esempio 5.1.1.** Dato l'attributo composto

$$[\text{Via} : \text{string}, \text{Numero} : \text{int}, \text{Citta} : \text{string}]$$

otteniamo



# 6 Algebra relazionale

L'**algebra relazionale** è l'insieme degli operatori su relazioni che danno come risultato relazioni. Viene usato come rappresentazione interna delle interrogazioni e non come linguaggio di interrogazione dei DBMS.
Il **calcolo relazionale** è invece il linguaggio dichiarativo di tipo logico dal quale è stato derivato SQL.

## 6.1 Notazione

**6.1.0.1 Nomi di relazioni** $R, S, \ldots$

**6.1.0.2 Nomi di attributi** $A, B, C, A_1, A_2, \ldots$

**6.1.0.3 Insiemi di attributi** $X, Y, X_1, \ldots$

**6.1.0.4 Unione di insiemi di attributi** $X \cup Y = XY$

**6.1.0.5 Relazione con ennuple** Date le ennuple $t_1, t_2, \ldots, t_n$ la relazione è denotata con $\{t_1, t_2, \ldots, t_n\}$

**6.1.0.6 Relazione vuota** $\{\}$

**6.1.0.7 Valore attributo** Data la ennupla $t_k$, il suo valore $A_i$ è denotato da $t_k.A_i$

**6.1.0.8 Ennupla specifica ad attributi** Se $X$ è sottoinsieme degli attributi di $t$, $t.X$ o $t[X]$ allora denota l'ennupla ottenuta da $t$ considerando solo gli attributi in $X$

**6.1.0.9 Ambiguità** Se $R$ ed $S$ hanno lo stesso attributo $A_j$, in caso di ambiguità, $R.A_j$ denota l'attributo $A_j$ della relazione $R$ ed $S.A_j$ denota l'attributo $A_j$ della relazione $S$.

## 6.2 Operatori primitivi

### 6.2.1 Ridenominazione

Viene utilizzato per cambiare il nome di una relazione e di conseguenza anche il suo **tipo**.

$$\rho_{A \leftarrow B}(R) \tag{1}$$

$$T' = \rho_{A \leftarrow A'}(T) \tag{2}$$

| **Id** | **Nome** | **Età** |
|---|---|---|
| 7274 | Rossi | 42 |
| 7432 | Neri | 54 |
| 9824 | Verdi | 45 |

$\longrightarrow$

| **Matricola** | **Nome** | **Età** |
|---|---|---|
| 7274 | Rossi | 42 |
| 7432 | Neri | 54 |
| 9824 | Verdi | 45 |

Table 1: Laureati

### 6.2.2 Unione

Restituisce la relazione ottenuta facendo l'unione delle ennuple di $R$ con quelle di $S$ dove $R$ e $S$ sono relazioni dello stesso tipo.

$$R \cup S \tag{3}$$

| **A** | **B** | **C** |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b1 | c2 |
| a2 | b1 | c1 |
| a3 | b1 | c1 |

$\cup$

| **A** | **B** | **C** |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b2 | c2 |

$\longrightarrow$

| **A** | **B** | **C** |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b1 | c2 |
| a1 | b2 | c2 |
| a2 | b1 | c1 |
| a3 | b1 | c1 |

### 6.2.3 Differenza

Restituisce la relazione contenente le ennuple di $R$ non presenti in $S$.

$$R - S \tag{4}$$

| **A** | **B** | **C** |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b1 | c2 |
| a2 | b1 | c1 |
| a3 | b1 | c1 |

$-$

| **A** | **B** | **C** |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b2 | c2 |

$\longrightarrow$

| **A** | **B** | **C** |
|---|---|---|
| a1 | b1 | c2 |
| a2 | b1 | c1 |
| a3 | b1 | c1 |

### 6.2.4 Proiezione

Restituisce una relazione i cui elementi sono la copia delle ennuple di $R$ proiettate (ristrette) sugli attributi $A_1, \ldots, A_n$. Eventuali ennuple che dopo la proiezione sono uguali appaiono solo una volta.

$$\pi_{A_1,\ldots,A_n}(R) \tag{5}$$

Data la tabella usata negli esempi precedenti, la proiezione è:

$\pi_A(R) =$

| **A** |
|---|
| a1 |
| a2 |
| a3 |

**6.2.4.1 Cardinalità** Una proiezione conterrà al più tante ennuple quanto l'operando:

- Se $X$ è una **superchiave** di $R$, allora $\pi_X(R)$ contiene esattamente tante ennuple quante $R$
- **Altrimenti** potrebbero esistere valori ripetuti su quegli attributi, che quindi vengono rappresentati una sola volta

### 6.2.5 Restrizione o selezione

Restituisce una relazione dello stesso tipo (schema) di $R$ i cui elementi sono la copia delle ennuple di $R$ (un sottoinsieme) che soddisfano la condizione $\phi$ definita come:

- $A_i \phi A_j$ con $A_i$ e $A_j$ attributi di $R$ e $\theta$ un operatore di confronto $\{<, >, =, \neq, \leq, \geq\}$
- $A_i \theta c$ oppure $c \theta A_i$, con $\theta$ operatore di confronto e $c$ costante nel dominio di $A_i$
- Se $\phi$ e $\psi$ sono formule, allora lo sono anche $\phi \wedge \psi$, $\phi \vee \psi$ e $\neg\psi$

$$\sigma_\phi(R) \tag{6}$$

Data la tabella usata negli esempi precedenti abbiamo

$\sigma_{A=a_1}(R) =$

| **A** | **B** | **C** |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b1 | c2 |

**Osservazione 6.2.1** (Valori nulli)**.** La presenza di valori **nulli** negli attributi usati per la restrizione portano all'assenza di **atomicità** nell'operazione.

$$\sigma_{\text{Età}>30}(\text{Persone}) \cup \sigma_{\text{Età}\leq 30}(\text{Persone}) \neq \text{Persone}$$

Per mantenerla è quindi necessario utilizzare i costrutti *IS NULL* e *IS NOT NULL*:

$$\sigma_{\text{Età}>30}(\text{Persone}) \cup \sigma_{\text{Età}\leq 30}(\text{Persone}) \cup \sigma_{\text{Età IS NULL}}(\text{Persone}) = \text{Persone}$$

### 6.2.6 Prodotto

Con $R$ e $S$ relazioni con attributi distinti, il loro prodotto è una relazione con elementi ottenuti concatenando ogni ennupla di $R$ con ogni ennupla di $S$. La relazione risultante ha grado uguale alla somma dei gradi degli operandi e cardinalità uguale al prodotto delle cardinalità degli operandi.

$$R \times S \tag{7}$$

| **A** | **B** | **C** |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b1 | c2 |
| a2 | b1 | c1 |
| a3 | b1 | c1 |

$\times$

| **A'** | **D** |
|---|---|
| a1 | d1 |
| a2 | d2 |

$\longrightarrow$

| **A** | **B** | **C** | **A'** | **D** |
|---|---|---|---|---|
| a1 | b1 | c1 | a1 | d1 |
| a1 | b1 | c1 | a2 | d2 |
| a1 | b1 | c2 | a1 | d1 |
| a1 | b1 | c2 | a2 | d2 |
| a2 | b1 | c1 | a1 | d1 |
| a2 | b1 | c1 | a2 | d2 |
| a3 | b1 | c1 | a1 | d1 |
| a3 | b1 | c1 | a2 | d2 |

**Osservazione 6.2.2.** Il prodotto di due relazioni è un'operazione che di solito non si usa da sola. Pertanto per concatenare ennuple in associazione, si restringe sempre il prodotto alle ennuple con valore uguale della chiave esterna e chiave primaria. Per questa si introduce un operatore derivato, chiamato **giunzione**, per riferirsi alla combinazione di queste due operazioni.

## 6.3 Operatori derivati

### 6.3.1 Intersezione

Restituisce la relazione ottenuta facendo l'intersezione delle ennuple di $R$ con quelle di $S$ dove $R$ e $S$ sono relazioni dello stesso tipo.

$$R \cap S = R - (R - S) \tag{8}$$

| **A** | **B** | **C** |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b1 | c2 |
| a2 | b1 | c1 |
| a3 | b1 | c1 |

$\cap$

| **A** | **B** | **C** |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b2 | c2 |

$\longrightarrow$

| **A** | **B** | **C** |
|---|---|---|
| a1 | b1 | c1 |

### 6.3.2 Inner Join

Restituisce la relazione contenente le ennuple del prodotto cartesiano di $R \times S$ con valori uguali per gli attributi $A_i$ e $A_j$ dove $R$ e $S$ sono relazioni di tipo diverso (con attributi distinti) tra i quali c'è $A_i$ in $R$ e $A_j$ in $S$.

$$R \bowtie_{A_i=A_j} S = \sigma_{A_i=A_j}(R \times S) \tag{9}$$

Date le tabelle usate come esempio nel prodotto ($R$ e $T'$), abbiamo

$R \bowtie_{A=A'} T' =$

| A | B | C | A' | D |
|---|---|---|---|---|
| a1 | b1 | c1 | a1 | d1 |
| a1 | b1 | c2 | a1 | d1 |
| a2 | b1 | c1 | a2 | d2 |

### 6.3.3 Theta Join

È il caso più generale della *Inner Join*: qui la condizione $\theta$ è una congiunzione di termini logici di confronto ($>$, $<$, $=$, ...).

$$R \bowtie_\sigma S \tag{10}$$

### 6.3.4 Natural Join

È una abbreviazione dell'*Inner Join* applicata a relazioni in cui l'associazione fra le ennuple è descritta con la chiave esterna e la chiave primaria costituite da attributi uguali. Restituisce la relazione contenente le ennuple di $R \times S$ con gli attributi di uguali nomi in $R$ ed $S$ definiti sugli stessi domini.

$$R \bowtie S = R \bowtie_{A=A} S = \sigma_{A=A}(R \times S) \tag{11}$$

| **A** | **B** | **C** |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b1 | c2 |
| a2 | b1 | c1 |
| a3 | b1 | c1 |

$\bowtie$

| **A** | **D** |
|---|---|
| a1 | d1 |
| a2 | d2 |

$\longrightarrow$

| **A** | **B** | **C** | **D** |
|---|---|---|---|
| a1 | b1 | c1 | d1 |
| a1 | b1 | c2 | d1 |
| a2 | b1 | c1 | d2 |

*Note* 6.3.1. Si noti che:

- Se $R$ ed $S$ non hanno attributi comuni allora $R \bowtie S = R \times S$
- Se $R$ ed $S$ hanno lo stesso schema allora $R \bowtie S = R \cap S$

**6.3.4.1 Cardinalità** Date le relazioni $R_1(A, B)$ e $R_2(B, C)$:

- Il numero di ennuple della loro join è:

$$0 \leq |R_1 \bowtie R_2| \leq |R_1| \times |R_2|$$

- Se il loro join è **completo** allora contiene un numero di ennuple almeno uguale al massimo tra $|R_1|$ e $|R_2|$
- Se il join coinvolge una **chiave** $B$ di $R_2$ allora:

$$0 \leq |R_1 \bowtie R_2| \leq |R_1|$$

- Se il join coinvolge una **chiave** $B$ di $R_2$ e un **vincolo di integrità referenziale** tra gli attributi di $R_1$ e la chiave di $R_2$ allora:

$$|R_1 \bowtie R_2| = |R_1|$$

**Osservazione 6.3.1** (Natural Join e Proiezione)**.** Osserviamo che valgono due cose:

- Dati $R_1(X_1)$ e $R_2(X_2)$ vale

$$\pi_{X_1}(R_1 \bowtie R_2) \subseteq R_1 \tag{12}$$

- Dati $R(X)$ con $X = X_1 \cup X_2$ vale

$$R \subseteq (\pi_{X_1}(R)) \bowtie (\pi_{X_2}(R)) \tag{13}$$

### 6.3.5 Semi Join

Restituisce le ennuple di $R$ che partecipano alla giunzione naturale di $R$ ed $S$ con $A_1, A_2, \ldots, A_m$ attributi di $R$.

$$RS = \pi_{A_1,A_2,\ldots,A_m}(R \bowtie S) \tag{14}$$

Considerando l'esempio precedente abbiamo

$RT =$

| A | B | C |
|---|---|---|
| a1 | b1 | c1 |
| a1 | b1 | c2 |
| a2 | b1 | c1 |

### 6.3.6 External Join

Il join esterno estende con valori nulli le ennuple che verrebbero tagliate fuori da un join interno. Ne esistono tre versioni:

- **Left**: mantiene tutte le ennuple del primo operando, estendendole con valori nulli se necessario

$$R \overset{\leftarrow}{\bowtie} S \tag{15}$$

- **Right**: come left ma con il secondo operando

$$R \overset{\rightarrow}{\bowtie} S \tag{16}$$

- **Full**: come left ma con entrambi gli operandi

$$R \overset{\leftrightarrow}{\bowtie} S \tag{17}$$

### 6.3.7 Self Join

Questa operazione è utilizzata quando va fatta una join di una tabella con se stessa. Spesso è importante fare anche delle ridenominazioni.

**Esempio 6.3.1** (Nonno nipote)**.** Data la relazione effettuiamo le seguenti operazioni

| Genitore | Figlio |
|---|---|
| Luca | Anna |
| Maria | Anna |
| Giorgio | Luca |
| Silvia | Maria |
| Enzo | Maria |

$$\rho_{\text{Genitore,Figlio}\leftarrow\text{Nonno,Genitore}}(\text{Genitore}) \bowtie \rho_{\text{Figlio}\leftarrow\text{Nipote}}(\text{Genitore})$$

| Nonno | Genitore |
|---|---|
| Luca | Anna |
| Maria | Anna |
| Giorgio | Luca |
| Silvia | Maria |
| Enzo | Maria |

$\bowtie$

| Genitore | Nipote |
|---|---|
| Luca | Anna |
| Maria | Anna |
| Giorgio | Luca |
| Silvia | Maria |
| Enzo | Maria |

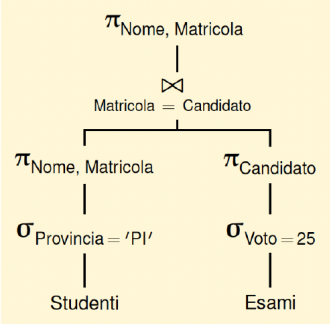
$=$

| Nonno | Genitore | Nipote |
|---|---|---|
| Giorgio | Luca | Anna |
| Silvia | Maria | Anna |
| Enzo | Maria | Anna |

## 6.4 Proprietà algebriche degli operatori

Un'espressione algebrica può essere rappresentata come un **albero**:

$$\pi_{\text{Nome,Matricola}}($$
$$\quad \pi_{\text{Nome,Matricola}}(\sigma_{\text{Provincia='PI'}}(\text{Studenti}))$$
$$\quad \bowtie_{\text{Matricola=Candidato}}$$
$$\quad \pi_{\text{Candidato}}(\sigma_{\text{Voto=25}}(\text{Esami}))$$
$$)$$




Un'espressione dell'algebra relazionale può essere trasformata in un'altra equivalente sfruttando alcune proprietà degli operatori. Questo aiuta a ridurre il costo di esecuzione. Le proprietà più utili sono quelle che permettono di anticipare la **restrizione** e la **proiezione**:

- **Raggruppamento** di **restrizioni**

$$\sigma_{C_1}(\sigma_{C_2}(R)) \equiv \sigma_{C_1 \wedge C_2}(R)$$

- **Raggruppamento** di **proiezioni**

$$\sigma_{C_1 \wedge C_2}(R \times S) \equiv \sigma_{C_1}(R) \times \sigma_{C_2}(S)$$

- **Commutatività** della **restrizione**, **proiezione**, **prodotto**, **giunzione** e degli **operatori insiemistici**

$$(R \times S) \equiv (S \times R)$$

- **Anticipazione** della **restrizione** rispetto al prodotto e alla giunzione e rispetto agli operatori insiemistici
- **Anticipazione** della **proiezione** rispetto al prodotto e alla giunzione e rispetto agli operatori insiemistici
- Eliminazioni di proiezioni superflue

$$\pi_A(\pi_{A,B}(R)) \equiv \pi_A(R)$$

**Osservazione 6.4.1** (Non distributività della proiezione rispetto alla differenza)**.** In generale vale che

$$\pi_A(R_1 - R_2) \neq \pi_A(R_1) - \pi_A(R_2) \tag{18}$$

## 6.5 Quantificatori

Esistono tre tipi di quantificatori:

- **Esistenziale**: e.g. tutti gli studenti che hanno preso *almeno* un voto maggiore di 28
- **Differenza**: e.g. tutti gli studenti che NON hanno mai preso un voto maggiore di 28
- **Universale**: e.g tutti gli studenti che hanno preso SOLO voti maggiori di 28

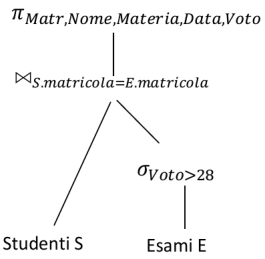



Figure 1: Esistenziale

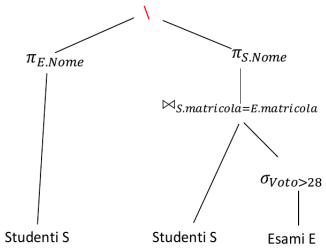



Figure 2: Differenza

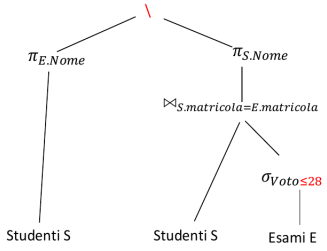



Figure 3: Universale

## 6.6 Operatori non insiemistici

### 6.6.1 Group By

Data una relazione $R$, i suoi attributi $A_i$ e le espressioni che usano funzioni di aggregazione $f_i$ (min, max, count, sum, ...), definiamo il raggruppamento come una relazione calcolata:

1. Partizionando le ennuple di $R$ mettendo nello stesso gruppo tutte le ennuple con valori uguali degli $A_i$
2. Si calcolano le espressioni $f_i$ per ogni gruppo
3. Per ogni gruppo si restituisce una sola ennupla con come attributi i valori degli $A_i$ e delle espressioni $f_i$

$$_{\{A_i\}}\gamma_{\{f_i\}}(R) \tag{19}$$

Il raggruppamento gode dell'**anticipazione** rispetto alla proiezione:

$$\sigma_C(_{X}\gamma_{F}(R)) \equiv {}_{X}\gamma_{F}(\sigma_C(R)) \tag{20}$$

**Esempio 6.6.1.** Vogliamo ottenere per ogni candidato il numero degli esami, il voto minimo, massimo e medio:

$$_{\{\text{Candidato}\}}\gamma_{\{\text{count}(*),\text{min}(\text{Voto}),\text{max}(\text{Voto}),\text{avg}(\text{Voto})\}}(\text{Esami})$$

| Materia | Candidato | Voto | Docente |
|---|---|---|---|
| DA | 1 | 20 | 10 |
| LFC | 2 | 30 | 20 |
| MTI | 1 | 30 | 30 |
| LP | 2 | 20 | 40 |

↓

| Candidato | Count(*) | min(Voto) | max(Voto) | avg(Voto) |
|---|---|---|---|---|
| 1 | 2 | 20 | 24 | 22 |
| 2 | 2 | 20 | 30 | 25 |

### 6.6.2 Proiezione generalizzata

Estende la proiezione con la possibilità di usare costanti o espressioni aritmetiche nella lista degli attributi. Permette anche di aggiungere **etichette** alle espressioni tramite l'operatore **AS**:

$$\pi_{e_1 \text{ AS ide}_1, e_2 \text{ AS ide}_2, \ldots}(R) \tag{21}$$

Dove $e_1, e_2, \ldots$ sono espressioni aritmetiche ottenute a partire da costanti e $\text{ide}_1, \text{ide}_2, \ldots$ sono etichette distinte.

### 6.6.3 Proiezione multi-insiemistica

Funziona come la proiezione ma senza eliminazione dei duplicati. Restituisce dei multi-insiemi e quindi si possono utilizzare solo come radice di un albero logico.

$$\pi^b_{\{A_i\}}(R) \tag{22}$$

### 6.6.4 Ordinamento

Ordina tutte le ennuple di $R$ in ordine crescente o decrescente rispetto agli attributi $A_i$.

$$\tau_{\{A_i\}}(R) \tag{23}$$

## 6.7 Calcolo relazionale

Il calcolo relazionale è un linguaggio che permette di definire il risultato di un'interrogazione (**query**) come l'insieme di quelle ennuple che soddisfano una certa condizione $\phi$.

L'algebra dà la possibilità di scrivere espressioni in cui gli operatori sono applicati al risultato di altri operatori (espressioni annidate). Il calcolo ha una **struttura piatta** ma permette di esprimere condizioni più **complesse**.
Un linguaggio che si colloca a metà tra i due stili si può ottenere:

1. Aggiungendo al calcolo la possibilità di annidare il costruttore di insiemi
2. Aggiungendo all'algebra la possibilità di avere nell'operatore di restrizione condizioni che fanno uso anche di quantificatori e di predicati di appartenenza.

Il risultato è un linguaggio che ha sia la capacità di esprimere interrogazioni in modo annidato che la possibilità di esprimere condizioni logiche complesse, come accade nel linguaggio SQL.

**Esempio 6.7.1.** L'insieme delle matricole degli Studenti che hanno superato qualcuno degli esami elencati nella relazione Materie, si può definire come

$$\{t.\text{Matricola} | t \in \text{Studenti}, \exists m \in \text{Materie}. \exists e \in \text{ProveEsami}.$$
$$e.\text{Candidato} = t.\text{Matricola} \wedge e.\text{Materia} = m.\text{Materia}\}$$

che equivale a

$$\pi_{\text{Matricola}}(\text{Studenti} \underset{\text{Matricola=Candidato}}{\bowtie} (\text{ProveEsami} \bowtie \text{Materie}))$$

# 7 Data Definition Language

## 7.1 Viste

Le **Viste Logiche** possono essere definite come delle tabelle **virtuali**, i cui dati sono riaggregazioni dei dati contenuti nelle tabelle fisiche, senza contenerli effettivamente.
Le viste permettono di **semplificare** la rappresentazione dei dati ed evitare di ripetere query molto complesse. Forniscono inoltre un'ulteriore **sicurezza** in quanto si può fornire l'accesso di un utente solo ad esse e non a tutta la BD.
Hanno però alcune limitazioni:

- Non è consentito usare **ORDER BY**
- A seconda del DBMS:
  - Non si possono usare **UNION**, **INTERSECT** e **EXCEPT**
  - **INTERSECT** e **EXCEPT** si possono realizzare con la **SELECT**

### 7.1.1 Creazione

Si creano tramite il seguente comando:

```
CREATE VIEW NomeVista [ ( ListaAttributi ) ] AS SelectSQL [ with [ local | cascaded ] check option ]
```

Si può scegliere di specificare i nuovi nomi delle colonne in **ListaAttributi**, altrimenti assumeranno gli stessi della tabella.

**Osservazione 7.1.1** (Viste di gruppo)**.** Una vista di gruppo è una vista in cui una delle colonne è una funzione di gruppo. In questo caso è obbligatorio assegnare un nome alla colonna della vista corrispondente alla funzione.

### 7.1.2 Modifica

Mentre il **contenuto** della vista è dinamico, la sua **struttura** è statica, quindi se viene aggiunta una colonna, questa non viene estesa alla vista.
Perché una vista sia **aggiornabile** (non strutturalmente ma a livello di contenuto), deve esistere una corrispondenza **biunivoca** tra le sue righe e quelle della tabella, ovvero:

- **SELECT** senza **DISTINCT** e solo di attributi
- **FROM** una sola tabella modificabile
- **WHERE** senza SottoSelect
- Assenza di **GROUP BY** e **HAVING**

L'aggiornamento può essere utile nel caso in cui si vuole che degli utenti senza tutti i privilegi possano comunque fare modifiche limitate (e.g. l'amministrazione che può modificare il numero di telefono di un cliente). Per aggiornare si usa:

```
INSERT INTO NomeVista (ListaAttributi) VALUES (ListaValori)
```

#### 7.1.2.1 Controllo dell'aggiornamento

Se si prevede che la view sia modificabile, alla sua creazione va aggiunto **WITH CHECK OPTION**. Questo garantisce che eventuali inserimenti saranno permessi solo se soddisfano la clausola **WHERE** specificata alla creazione. **LOCAL** e **CASCADE** consentono di decidere nel caso in cui la vista sia creata sulla base di un'altra, se è necessario controllare o meno tutte le clausole **WHERE**.

### 7.1.3 Eliminazione

Viene utilizzato il seguente comando

```
DROP VIEW nome_view {RESTRICT/CASCADE}
```

Dove **RESTRICT** indica che la view viene eliminata solo se non è riferita da altri oggetti mentre **CASCADE** elimina anche tutte le altre dipendenze da essa.

## 7.2 Trigger

Un trigger definisce un'azione che il database deve attivare automaticamente quando si verifica un determinato **evento** nella BD, ovvero determinati comandi quali:

- Comandi **DML** quali INSERT, UPDATE e DELETE
- Negli ultimi DBMS anche comandi **DDL** come CREATE VIEW
- Aggiornamenti di specifiche colonne

Un trigger può essere:

- **Attivo** se modifica lo stato della BD
- **Passivo** se provoca solo il fallimento della transazione in certi casi

### 7.2.1 Granularità

I trigger possono essere eseguiti su due livelli:

- **Riga**: vengono eseguiti una volta per ogni riga modificata nella transazione. Spesso utilizzati per **audit** dei dati e per **sincronizzazione**. Va specificato con

  ```
  FOR EACH ROW
  ```

- **Istruzione**: vengono eseguiti una volta per transazione. Sono usati Per attività correlate ai dati (e.g. sicurezza)

### 7.2.2 Creazione

La struttura di un comando di creazione di un trigger è la seguente:

```
CREATE TRIGGER Nome
BEFORE/AFTER INSERT/DELETE/UPDATE of ATTRIBUTI {, Evento}
ON Tabella [WHEN Condizione]
FOR EACH {ROW/STATEMENT}
Azione
```

### 7.2.3 INSTEAD OF

Questo comando può essere usato per eseguire un'azione diversa invece di quella che dovrebbe succedere con l'evento previsto. Può essere usato anche sulle viste ma deve per forza essere a livello di riga.

## 7.3 Accessi

Ogni **risorsa** dello schema può essere protetta dal creatore della risorsa. Il creatore della DB, salvo sue diverse specifiche, è l'unico a poter eseguire **CREATE**, **ALTER** e **DROP** ed è l'unico a poter garantire o rimuovere privilegi.

### 7.3.1 Concessione

```
GRANT Privilegi ON Oggetto TO Utenti [ WITH GRANT OPTION ]
```

**WITH GRANT OPTION** specifica se il privilegio può essere trasmesso o meno ad altri utenti.

### 7.3.2 Revoca

```
REVOKE [ GRANT OPTION FOR ] Privileges ON Resource FROM Users [ RESTRICT | CASCADE ]
```

Il comando revoca quei privilegi anche a chiunque li abbia ricevuti da quell'utente. **RESTRICT** indica che il comando non deve essere eseguito se comporterebbe la revoca di qualcos'altro mentre **CASCADE** ne forza l'esecuzione.

### 7.3.3 Grafo autorizzazioni

È il grafo che rappresenta quali autorizzazioni sono state concesse e da chi. Se un nodo $N$ ha un arco uscente con un privilegio, allora esiste un cammino da SYSTEM a $N$ con ogni arco etichettato dallo stesso privilegio WITH GRANT OPTION.

**Esempio 7.3.1.** Ogni utente ($A$, $B$, $C$, $I$) esegue una serie di comandi:

```
I: GRANT SELECT ON R TO A WITH GRANT OPTION
A: GRANT SELECT ON R TO B WITH GRANT OPTION
B: GRANT SELECT ON R TO A WITH GRANT OPTION
I: GRANT SELECT ON R TO C WITH GRANT OPTION
C: GRANT SELECT ON R TO B WITH GRANT OPTION

I: REVOKE SELECT ON R FROM A CASCADE

I: REVOKE SELECT ON R FROM C CASCADE
```

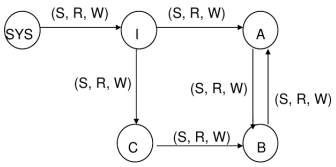



Figure 4: GRANT

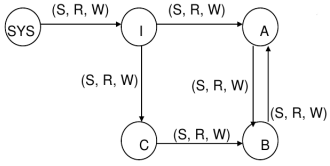



Figure 5: Prima REVOKE

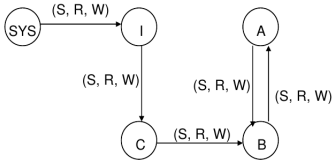



Figure 6: Seconda REVOKE

## 7.4 Indici

Gli Indici sono strutture dati che vengono create su tabelle per eseguire alcune query più velocemente. Non essendo un comando standard può avere varie sintassi:

```
CREATE INDEX NomeIdx ON Tabella(Attributi)
CREATE INDEX NomeIdx ON Tabella WITH STRUCTURE = BTREE, KEY = (Attributi)
DROP INDEX NomeIdx
```

## 7.5 Catalogo dei metadati

Il catalogo dei metadati consiste in un insieme di tabelle di sistema. Alcuni esempi sono:

- Delle **password**

```
PASSWORD(username, password)
```

- Delle **BD**

```
SYSDB(dbname, creator, dbpath, remarks)
```

- Delle **tabelle** e **view** (type)

```
SYSTABLES(name, creator, type, colcount, filename, remarks)
```

- Degli **attributi**

```
SYSCOLUMNS(name, tbname, tbcreator, colno, coltype, lenght, default, remarks)
```

- Degli **indici**

```
SYSINDEXES(name, tbname, creator, uniquerule, colcount)
```

# 8 Normalizzazione

Dati diversi modelli relazionali e diverse possibili rappresentazioni sorge il problema di verificare se:

- queste diverse rappresentazioni sono tra di loro **equivalenti**
- queste rappresentazioni sono di **buona qualità** (assenza di **anomalie**)

**Definizione 8.0.1** (Teoria della normalizzazione)**.** *La teoria della normalizzazione si occupa di **definire criteri formali** per giudicare l'**equivalenza** di schemi e la **qualità** di tali schemi, e di definire algoritmi per trasformare uno schema in un altro equivalente ma privo di anomalie.*

## 8.1 Linee guida

Ci sono quattro principali linee guida per avere una corretta progettazione:

- **Semantica degli attributi**: si devono progettare schemi relazionali in modo tale che sia semplice spiegarne il significato evitando di unire attributi provenienti da più tipi di classi e tipi di associazione in una unica relazione.
- **Ridondanza**: si devono progettare schemi relazionali in modo che nelle relazioni non siano presenti anomalie di inserimento, cancellazione o modifica. Se sono presenti (e le si vuole mantenere), le si rilevi e ci si assicuri che i programmi che aggiornano la BD operino correttamente
- **Valori nulli**: evitare di porre in relazione di base attributi i cui valori possono essere spesso nulli. Se è inevitabile, assicurarsi che essi si presentino solo in casi eccezionali e che non riguardino una maggioranza di tuple nella relazione.
- **Tuple spurie**: si devono progettare schemi relazionali in modo tale che essi possano essere riuniti tramite giunzioni con condizioni di uguaglianza su attributi che sono o chiavi primarie o chiavi esterne in modo da garantire che non vengano generate tuple spurie. Evitare relazioni che contengono attributi di giunzione diversi dalle combinazioni chiave esterna-chiave primaria.

### 8.1.1 Anomalie

Le principali anomalie che si trovano sono:

- **Ridondanze**
- Potenziali **inconsistenze**
- Anomalie nelle **inserzioni**
- Anomalie nelle **eliminazioni**

**Esempio 8.1.1.** Supponiamo di avere una BD di una biblioteca

**Biblioteca**

| NomeUtente | Residenza | Telefono | NumeroLibro | Autore | Titolo | DataPrestito |
|---|---|---|---|---|---|---|
| Rossi Carlo | Carrara | 75444 | XY188A | Boccaccio | Decameron | 07/07/19 |
| Paolicchi Luca | Avenza | 59729 | XY256B | Verga | Novelle | 07/07/19 |
| Pastine Maurizio | Dogana | 66133 | XY090C | Petrarca | Canzoniere | 01/08/19 |
| Paolicchi Laura | Avenza | 59729 | XY101A | Dante | Vita Nova | 05/08/19 |
| Paolicchi Luca | Avenza | 59729 | XY701B | Manzoni | Adelchi | 14/01/20 |
| Paolicchi Luca | Avenza | 59729 | XY008C | Moravia | La noia | 17/08/20 |

Questo schema presenta due anomalie:

- **Ridondanza** delle informazioni personali di un utente
- Impossibilità di rappresentare le informazioni sugli utenti della biblioteca che non hanno preso in prestito libri

Una possibile soluzione è la **decomposizione** in due relazioni:

```
Utenti(NomeUtente, Residenza, Telefono)
Prestiti(NumeroLibro, Autore, Titolo, Data, NomeUtente*)
```

Non è la soluzione migliore per costo di operazioni ma funziona.

### 8.1.2 Obiettivi

L'obiettivo della normalizzazione è fornire **strumenti formali** per la progettazione di basi di dati che non presentino anomalie senza prendere in considerazione il costo delle operazioni. In particolare si occupa di:

- **Equivalenza di schemi**: definire quando due schemi sono equivalenti
- **Qualità degli schemi**: definire quando uno schema è migliore di un altro
- **Trasformazione degli schemi**: trovare metodi algoritmici per ottenere da uno schema uno migliore ed equivalente

### 8.1.3 Schema di relazione universale

**Definizione 8.1.1** (Schema di relazione universale)**.** *Assumendo come **ipotesi** che tutti i fatti sono descritti da attributi di un'unica relazione (relazione universale), cioè gli attributi hanno un significato globale.*
***Definiamo** lo schema di relazione universale U come di una base di dati relazionale ha come attributi l'unione degli attributi di tutte le relazioni della base di dati.*

### 8.1.4 Notazione

Di seguito la notazione di base:

- **Singoli attributi**: $A, B, C, A_1, A_2, \ldots$
- **Insiemi di attributi**: $T, X, Y, X_1, \ldots$
- **Abbreviazioni**:
  - $XY \equiv X \cup Y$
  - $AB \equiv \{A, B\}$
  - $A_1, A_2, \ldots, A_n \equiv \{A_1, A_2, \ldots, A_n\}$
  - $XA \equiv A \cup \{A\}$
- **Schema di relazione**: $R(T)$, $r$ la sua **istanza** e $t$ l'**ennupla** dell'istanza
- Se $X \subseteq T$ allora $t[X]$ indica l'**ennupla** ottenuta da $T$ considerando solo gli attributi in $X$

## 8.2 Dipendenze funzionali

Informalmente, una dipendenza funzionale indica che dato un insieme di attributi, questi ne determinano in maniera univoca altri.

**Definizione 8.2.1** (Dipendenza funzionale)**.** *Dato uno schema $R(T)$ e $X, Y \subseteq T$, una dipendenza funzionale è un vincolo su $R$ del tipo $X \to Y$ se:*

- $\forall r$ *istanza valida di* $R$
- $\forall t_1, t_2 \in r. t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$

**Esempio 8.2.1.** Data la seguente tabella esiste la dipendenza funzionale Matricola → Cognome

| Matricola | Cognome |
|---|---|
| 1 | Rossi |
| 2 | Verdi |
| 3 | Rossi |
| 4 | Viola |

*Note* 8.2.1. Essendo definite solo all'interno di una relazione non possono esistere fra attributi di relazioni diverse.

*Note* 8.2.2. Sono proprietà **intensionali**, quindi legate al significato dei fatti. Non è possibile dedurle dall'osservazione di alcune istanze.

**Definizione 8.2.2** (Dipendenza funzionale atomica)**.** *Ogni DF del tipo $X \to A_1, A_2, \ldots, A_n$ si può scomporre in $X \to A_1, X \to A_2, \ldots, X \to A_n$. DF del tipo $X \to A$ sono dette **atomiche**.*

**Esempio 8.2.2.** Dato lo schema

$$\text{DotazioniLibri}(\text{CodiceLibro}, \text{NomeNegozio}, \text{IndNegozio}, \text{Titolo}, \text{Quantità})$$

la dipendenza funzionale

$$\text{CodiceLibro}, \text{NomeNegozio} \to \text{IndNegozio}, \text{Titolo}, \text{Quantità}$$

si può scomporre in

$$\begin{aligned}&\text{CodiceLibro}, \text{NomeNegozio} \to \text{IndNegozio}\\ &\text{CodiceLibro}, \text{NomeNegozio} \to \text{Titolo}\\ &\text{CodiceLibro}, \text{NomeNegozio} \to \text{Quantità}\end{aligned}$$

**Definizione 8.2.3** (Dipendenza funzionale banale)**.** *Una DF $X \to A$ è detta **banale** se $A \in X$.*

### 8.2.1 Chiavi

Le dipendenze funzionali sono una generalizzazione del vincolo di chiave e superchiave. Infatti, dato uno schema $R(T)$, $X, Y \subseteq T$ e $r$ istanza di $R$, se $r \models X \to Y$ allora se $X$

- **Non** è **superchiave** allora ho un'**anomalia**
- È **superchiave**, allora $\forall r.r \models X \to T$
- È **chiave**, allora $\forall r.r \models X \to T$ e $X \to T$ è una DF **completa**

### 8.2.2 Utilizzo

Le dipendenze funzionali vengono utilizzate per specificare il significato dei fatti rappresentati in uno schema trovando eventuali anomalie e normalizzandolo. Per questo motivo da ora saranno incluse nella definizione:

$$R(T, F) \tag{24}$$

**8.2.2.1 Dipendenze derivate** Dato uno schema $R(T, F)$, le sue istanza soddisfano le DF e anche quelle derivabili da esse. Ad esempio dato

$$R(T, F = \{X \to Y, X \to Z\}) \qquad X, Y, Z \subseteq T, W \subseteq X$$

allora anche $X \to W$ e $X \to YZ$ saranno soddisfatte:

- la prima è ovvia in quanto $W$ è sottoinsieme di $X$
- se $t_1[X] = t_2[X]$ allora $t_1[Y] = t_2[Y] \wedge t_1[Z] = t_2[Z]$ e quindi $t_1[YZ] = t_2[YZ]$

**Definizione 8.2.4** (Implicazione di dipendenze)**.** *Dato $R(T)$ e dato $F$, diciamo che $F \models X \to Y$ ($F$ implica logicamente $X \to Y$), se ogni istanza $r$ di $R(T)$ che soddisfa $F$ soddisfa anche $X \to Y$.*

**8.2.2.2 Assiomatizzazione** Sia $RI$ un insieme di regole di inferenze per $F$, ovvero per derivare altre DF a partire da $F$. Indichiamo con $F \vdash X \to Y$ il fatto che $X \to Y$ sia derivabile da $F$ usando $RI$. L'insieme $RI$ è:

- **Corretto**: se $X \to Y$ è derivabile da $F$ allora ogni istanza che soddisfa $F$ soddisfa anche $X \to Y$

$$F \vdash X \to Y \implies F \models X \to Y$$

- **Completo**: se ogni istanza che soddisfa $F$ soddisfa anche $X \to Y$ implica che $X \to Y$ è derivabile da $F$

$$F \models X \to Y \implies F \vdash X \to Y$$

**8.2.2.3 Regole di inferenza** Gli assiomi di Armstrong (1974) sono il più noto insieme **corretto** e **completo** di regole di inferenza per DF:

- **Riflessività**

$$Y \subseteq X \implies X \to Y \tag{25}$$

- **Arricchimento**

$$X \to Y, Z \subseteq T \implies XZ \to YZ \tag{26}$$

- **Transitività**

$$X \to Y, Y \to Z \implies X \to Z \tag{27}$$

**Definizione 8.2.5** (Derivazione)**.** *Una **derivazione** di $f$ da $F$ è una sequenza finita $f_1, \ldots, f_m$ di dipendenze dove $f_m = f$ e ogni $f_i$ è un elemento di $F$ oppure è ottenuta dalle precedenti dipendenze delle derivazione $f_1, \ldots, f_{i-1}$ usando regole di inferenza.*
*Una **sottosequenza** $f_1, \ldots, f_k$ per una derivazione $f_1, \ldots, f_m$ è anch'essa una derivazione, quindi $F \vdash f_k \forall k = 1, \ldots, m$.*

Sia $F$ un insieme di DF, diremo che $X \to Y$ sia **derivabile** da $F$ ($F \vdash X \to Y$), se $X \to Y$ può essere inferito da $F$ usando gli assiomi di Armstrong.
Vediamo altre regole di derivazione ottenute dagli assiomi di Armstrong:

- **Unione**: $\{X \to Y, X \to Z\} \vdash X \to YZ$
  1. $X \to Y$ per ipotesi
  2. $X \to XY$ per arricchimento da 1
  3. $X \to Z$ per ipotesi
  4. $XY \to YZ$ per arricchimento da 3
  5. $X \to YZ$ per transitività da 2, 4
- **Decomposizione**: $\{X \to YZ\} \vdash X \to Y$
  1. $X \to YZ$ per ipotesi
  2. $YZ \to Y$ per riflessività da $Y \subseteq YZ$
  3. $X \to Y$ per transitività da 1, 2
- **Indebolimento**: $\{X \to Y\} \vdash XZ \to Y$
  1. $X \to Y$ per ipotesi
  2. $XZ \to X$ per riflessività da $X \subseteq XZ$
  3. $XZ \to Y$ per transitività da 2, 1
- **Identità**: $\{\} \vdash X \to X$

**Teorema 8.2.1.** Gli assiomi di Armstrong sono **corretti** e **completi**.

*Proof.* Se una dipendenza è derivabile con gli assiomi di Armstrong allora è anche implicata logicamente (correttezza degli assiomi), e viceversa se una dipendenza è implicata logicamente allora è anche derivabile dagli assiomi (completezza degli assiomi).

- Correttezza
$$\forall f \qquad F \vdash f \Longrightarrow F \models f \tag{28}$$
- Completezza
$$\forall f \qquad F \models f \Longrightarrow F \vdash f \tag{29}$$

□

### 8.2.3 Chiusura

La **chiusura** è l'insieme di DF $X \to Y$ derivabili da $F$.

**Definizione 8.2.6** (Chiusura)**.** *Dato un insieme $F$ di DF, la sua chiusura, denotata con $F^+$ è*

$$F^+ = \{X \to Y | F \vdash X \to Y\} \tag{30}$$

La chiusura di $X$ rispetto ad $F$ è l'insieme di attributi determinati da $X$ tra quelli delle DF derivabili da $F$.

**Definizione 8.2.7** (Chiusura)**.** *Dato $R(T, F)$, e $X \subseteq T$, la chiusura di $X$ rispetto a $F$, denotata con $X_F^+$, (o $X^+$, se $F$ è chiaro dal contesto) è*

$$X_F^+ = \{A_i \in T | F \vdash X \to A_i\} \tag{31}$$

#### 8.2.3.1 Problema dell'implicazione

Il problema dell'implicazione consiste nel decidere se una DF $V \to W$ appartiene a $F^+$. La sua risoluzione con l'algoritmo di generare $F^+$ applicando ad $F$ ripetutamente gli assiomi di Armstrong ha una complessità esponenziale rispetto al numero di attributi dello schema.
Un algoritmo più semplice si basa sul seguente teorema:

**Teorema 8.2.2.** Da DF $X \to Y$ è derivabile da $F$ se e solo se $Y$ è sottoinsieme della chiusura di $X$ rispetto a $F$.

$$F \vdash X \to Y \Leftrightarrow Y \subseteq X_F^+. \tag{32}$$

```
input  R(T,F), X ⊆ T
output   X+
begin
  X+ = X
  while (X+ cambia) do
    for W → V in F with W ⊆ X+ and V ⊊ X+
      do X+ = X+ ∪ V
end
```

L'algoritmo **termina** perché gli attributi sono di numero finito e per dimostrare la **correttezza** si dimostra per induzione che $X_F^+ = X^+$.

#### 8.2.3.2 Definizione di chiavi

**Definizione 8.2.8** (Superchiave)**.** *Dato lo schema $R(T, F)$, un insieme di attributi $W \subseteq T$ è una superchiave di $R$ se $W \to T \in F^+$.*

**Definizione 8.2.9** (Chiave)**.** *Dato lo schema $R(T, F)$, un insieme di attributi $W \subseteq T$ è una chiave di $R$ se $W$ è una superchiave e non esiste un sottoinsieme stretto di $W$ che sia superchiave di $R$.*

**Definizione 8.2.10** (Attributo primo)**.** *Dato lo schema $R(T, F)$, un attributo $A \in T$ si dice attributo primo se e solo se appartiene ad almeno una chiave, altrimenti si dice non primo.*

**Trovare tutte le chiavi** L'algoritmo per trovare tutte le chiavi si basa su due proprietà:

1. Se un attributo $A$ di $T$ non appare a destra di alcuna dipendenza in $F$, allora $A$ appartiene ad ogni chiave di $R$ (altrimenti non può essere determinato)
2. Se un attributo $A$ di $T$ appare a destra di qualche dipendenza in $F$, ma non appare a sinistra di alcuna dipendenza non banale, allora $A$ non appartiene ad alcuna chiave.

Sia $X$ l'insieme degli attributi che non appaiono a destra di alcuna dipendenza in $F$. Da 1. segue che se $X^+ = T$, allora $X$ è una chiave di $R$ ed è anche l'unica possibile. Altrimenti, occorre aggiungere a $X$ altri attributi. Per 2. basta aggiungere gli attributi $W$ di $T$ che appaiono a destra di qualche dipendenza e a sinistra di qualche altra, uno alla volta evitando di aggiungere quelli che sono già in $X^+$ o quelli che producono un $X'$ che contiene una chiave già trovata.

```
input    R(T,F)
output   Insieme di tutte le chiavi

begin
  NoDes := T − ∪_{X→A∈F} A;
  SinDes := ∪_{X→A∈F} X ∩ ∪_{X→A∈F} A;
  Candidati := [NoDes::(SinDes)];
  Chiavi := [];
  while (Candidati non vuoto) do
    begin
      X::(Y) := first(Candidati);
      Candidati := rest(Candidati);
      if not some K in Chiavi with K ⊂ X
      then if X^+ = T then Chiavi := Chiavi + X;
        else begin
          A_1, ..., A_n := Y − X^+;
          for i in 1, ..., n do Candidati = Candidati.append(XA_i::(A_{i+1}, ..., A_n))
        end
    end
end
```

*Note* 8.2.3. Il problema di trovare tutte le chiavi di una relazione richiede un algoritmo di complessità **esponenziale**.

*Note* 8.2.4. Il problema di controllare se un attributo è primo è **NP-completo**.

### 8.2.4 Copertura canonica

**Definizione 8.2.11** (Copertura)**.** *Due insiemi di DF, $F$ e $G$, sullo schema $R$ sono equivalenti ($F \equiv G$) se e solo se $F^+ = G^+$. Se $F \equiv G$, allora $F$ è una copertura di $G$ e viceversa.*

**Definizione 8.2.12** (Attributo estraneo)**.** *Sia $F$ un insieme di DF. Data una $X \to Y \in F$, si dice che $X$ contiene un attributo estraneo $A_i$ se e solo se $(X - \{A_i\}) \to Y \in F^+$, cioè $F \vdash (X - \{A_i\}) \to Y$.*

**Esempio 8.2.3** (Attributo estraneo)**.** Sia

$$F = \{AB \to C, A \to B\}$$

Calcoliamo $A^+ = ABC$ e vediamo che $C$ dipende solo da $A$ e di conseguenza $B$ è **estraneo** in $AB \to C$.

**Definizione 8.2.13** (Dipendenza ridondante)**.** *$X \to Y$ è una dipendenza ridondante se e solo se $(F - \{X \to Y\})^+ = F^+$, equivalentemente $F - \{X \to Y\} \vdash X \to Y$.*

**Esempio 8.2.4** (Dipendenza ridondante)**.** Sia

$$F = \{B \to C, B \to A, C \to A\}$$

$B \to A$ è **ridondante** poiché possiamo dedurla da $B \to C$ e $C \to A$.

**Definizione 8.2.14** (Copertura canonica)**.** *$F$ è detta copertura canonica se e solo se:*

- *la parte destra di ogni DF in $F$ è un attributo, ovvero tutte le DF sono* ***atomiche***
- *non esistono* ***attributi estranei***
- *nessuna dipendenza in $F$ è* ***ridondante***

**Teorema 8.2.3** (Esistenza copertura canonica)**.** Per ogni insieme di dipendenze $F$ esiste una copertura canonica.

Per calcolare la copertura canonica si può applicare il seguente algoritmo:

1. Trasformare le dipendenze nella forma $X \to A$: si sostituisce l'insieme dato con quello equivalente che ha tutti i secondi membri costituiti da singoli attributi (dipendenze atomiche)
2. Eliminare gli attributi estranei: per ogni dipendenza si verifica se esistono attributi eliminabili dal primo membro
3. Eliminare le dipendenze ridondanti: per ogni dipendenza si verifica se può essere eliminata

## 8.3 Decomposizione di schemi

L'approccio da seguire per eliminare **anomalie** da uno schema mal definito, è quello di **decomporlo** in schemi più piccoli che godono di particolari proprietà (**forme normali**), ma sono in qualche senso equivalenti allo schema originale. L'intuizione è che si devono "estrarre" gli attributi che sono determinati da attributi non chiave ovvero "creare uno schema per ogni funzione".

Le **ridondanze** sui dati possono essere:

- **Concettuali**: non ci sono duplicazioni dello stesso dato ma sono memorizzate informazioni che possono essere ricavate da altre già contenute nella BD
- **Logiche**: esistono duplicazioni sui dati che possono generare anomalie

**Definizione 8.3.1** (Decomposizione)**.** *Dato uno schema $R(T)$, $\rho = \{R_1(T_1), \ldots, R_k(T_k)\}$ è una decomposizione di $R$ se e solo se $T_1 \cup \ldots \cup T_k = T$.*

**Esempio 8.3.1** (Decomposizione)**.** Prendiamo la relazione

```
Articoli(Kit, Componente, Tipo, QuantComp, PrezzoComp, Fornitore, PrezzoTot)
```

| Kit | Componente | Tipo | QuantComp | PrezzoComp | Fornitore | PrezzoTot |
|---|---|---|---|---|---|---|
| Libreria | Legno | Noce | 50 | 10 | A | 4400 |
| Libreria | Bulloni | Acciaio | 200 | 1 | B | 4400 |
| Libreria | Vetro | Cristallo | 3 | 50 | C | 4400 |
| Scaffale | Legno | Mogano | 37 | 15 | A | 555 |
| PC | Bulloni | Acciaio | 25 | 1 | B | 700 |
| PC | Tastiera | A3000 | 3 | 30 | D | 700 |
| PC | Mouse | B2000 | 5 | 45 | D | 700 |
| ... | ... | ... | ... | ... | ... | ... |

Identifichiamo le **ridondanze**:

- **PrezzoTot** è ripetuto in ogni tupla riferita allo stesso **Kit**
- **PrezzoComp** è ripetuto in ogni tupla che ha lo stesso valore di **Tipo** e **Fornitore**
- **Componente** è ripetuto in ogni tupla che ha lo stesso **Tipo**

Scriviamo poi le **dipendenze** derivate:

- **Tipo** → **Componente**
- **Kit** → **PrezzoTot**
- **Kit**, **Tipo** → **PrezzoComponente**, **QuantComp**, **Fornitore**

questo ci porta ad avere la seguente decomposizione

| Kit | Tipo | QuantComp | PrezzoComp | Fornitore |
|---|---|---|---|---|

| Kit | PrezzoTot |
|---|---|

| Tipo | Componente |
|---|---|

Una decomposizione dovrebbe sempre soddisfare le seguenti qualità:

- Decomposizione **senza perdita** che garantisce la ricostruzione delle informazioni originarie senza generazione di tuple spurie
- **Conservazione delle dipendenze** che garantisce il mantenimento dei vincoli di integrità (di dipendenza funzionale) originari

### 8.3.1 Conservazione dei dati

Per una decomposizione che preserva i dati, ogni istanza valida $r$ della relazione di partenza deve essere uguale alla giunzione naturale della sua proiezione sui $T_i$.

**Definizione 8.3.2** (Conservazione dei dati)**.** *$\rho = \{R_1(T_1), \ldots, R_k(T_k)\}$ è una decomposizione di uno schema $R(T)$ che preserva i dati se e solo se per ogni istanza valida $r$ di $R$:*

$$r = (\pi_{T1} r) \bowtie (\pi_{T_2} r) \bowtie \ldots \bowtie (\pi_{T_k} r) \tag{33}$$

**8.3.1.1 Perdita di informazione** Una **perdita di informazione** è quando, proiettando una relazione sui sottoschemi e poi facendo la giunzione, si ottengono più ennuple di quante ce ne fossero nella relazione originaria.

**Teorema 8.3.1** (Perdita di informazione)**.** Se $\rho = \{R_1(T_1), \ldots, R_k(T_k)\}$ è una decomposizione di $R(T)$, allora per ogni istanza $r$ di $R$

$$r \subseteq (\pi_{T_1} r) \bowtie (\pi_{T_2} r) \bowtie \ldots \bowtie (\pi_{T_k} r) \tag{34}$$

Uno schema $R(X)$ si decompone **senza perdite** negli schemi $R_1(X_1)$ ed $R_2(X_2)$ se, per ogni possibile istanza $r$ di $R(X)$, il join naturale delle proiezioni di $r$ su $X_1$ ed $X_2$ produce la tabella di partenza, cioè non contiene ennuple spurie.

$$_{X_1}(r) \bowtie {}_{X_2}(r) = r$$

**Esempio 8.3.2** (Perdita di informazione)**.** Supponiamo di avere lo schema

```
StudentiEdEsami(Matricola, Nome, Provincia, AnnoNascita, Materia, Voto)
```

e lo decomponiamo in

```
Studenti(Matricola, Nome, Provincia, AnnoNascita),
Esami(Nome, Materia, Voto)
```

Quando andiamo a fare la **giunzione naturale**, si creano tuple che prima non esistevano. Basta immaginare cosa succederebbe se due studenti con lo stesso nome avessero fatto esami diversi.

**Teorema 8.3.2** (Decomposizione senza perdita)**.** Se l'insieme degli attributi comuni alle due relazioni $(X1 \cap X2)$ è chiave per almeno una delle due relazioni decomposte allora la decomposizione è senza perdita.

*Proof.* Supponiamo $r$ sia una relazione sugli attributi $ABC$ e consideriamo le sue proiezioni $r_1$ su $AB$ e $r2$ su $AC$. Supponiamo che $r$ soddisfi la dipendenza funzionale $A \to C$. Allora $A$ è chiave per $r$ su $AC$ e quindi non ci sono in tale proiezione due tuple diverse sugli stessi valori di $A$. □

#### 8.3.1.2 Decomposizione binaria

**Teorema 8.3.3.** Sia $R(T, F)$ uno schema di relazione, la decomposizione $\rho = \{R_1(T_1), R_2(T_2)\}$ preserva i dati se e solo se $T_1 \cap T_2 \to T_1 \in F^+$ oppure $T_1 \cap T_2 \to T_2 \in F^+$.

### 8.3.2 Conservazione delle dipendenze

Una decomposizione preserva le dipendenze se ciascuna delle dipendenze funzionali dello schema originario coinvolge attributi che compaiono tutti insieme in uno degli schemi decomposti.

#### 8.3.2.1 Proiezione di una dipendenza funzionale

**Definizione 8.3.3** (Proiezione di una DF)**.** *Dato lo schema $R(T, F)$, e $T_1 \subseteq T$, la proiezione di $F$ su $T_1$ è $\pi_{T_1}(F) = \{X \to Y \in F^+ | XY \subseteq T_1\}$.*

Un algoritmo per il calcolo della proiezione è il seguente

```
for each Y ⊆ T1 do (Z := Y+; output Y → Z ∩ T1)
```

**Esempio 8.3.3** (Proiezione di DF)**.** Siano

$$R(A, B, C) \qquad F = \{A \to B, B \to C, C \to A\}$$

Abbiamo le seguenti proiezioni

$$\pi_{AB}(F) \equiv \{A \to B, B \to A\}$$
$$\pi_{AC}(F) \equiv \{A \to C, C \to A\}$$

**Definizione 8.3.4** (Conservazione delle dipendenze)**.** *Dato lo schema $R(T, F)$, la decomposizione $\rho = \{R_1, \ldots, R_n\}$ preserva le dipendenze se e solo se l'unione delle dipendenze in $\pi_{T_i}(F)$ è una copertura di $F$.*

*Note* 8.3.1. Il problema di stabilire se una decomposizione conserva le dipendenze è di complessità **polinomiale**.

**Teorema 8.3.4.** Sia $\rho = \{R_i(T_i, F_i)\}$ una decomposizione di $R(T, F)$ che preservi le dipendenze e tale che un $T_j$ sia una superchiave per $R$. Allora $\rho$ preserva i dati.

## 8.4 Forme normali

Una forma normale è una **proprietà** di una base di dati relazionale che ne garantisce la **qualità**, cioè l'assenza di determinati difetti.
Le forme normali sono:

- **Prima** forma normale: impone una restrizione sul tipo di una relazione: ogni attributo ha un tipo elementare
- **Seconda** forma normale: impone una restrizione sulle dipendenze
- **Terza** forma normale: impone una restrizione sulle dipendenze
- **BCNF**: la più naturale e restrittiva

### 8.4.1 Boyce-Cobb Normal Form

Una relazione $r$ è in BCNF se, per ogni DF non banale $X \to Y$ definita su di essa, $X$ contiene una chiave $K$ di $r$, ovvero $X$ è una **superchiave**. La forma normale richiede che i concetti in una relazione siano omogenei (solo proprietà direttamente associate alla chiave).

**Definizione 8.4.1** (BCNF)**.** *$R(T, F)$ è in BCNF se e solo se per ogni $X \to A \in F^+$ non banale ($A \notin X$), $X$ è una **superchiave**.*

**Teorema 8.4.1.** $R(T, F)$ è in BCNF se e solo se per ogni $X \to A \in F$ non banale, $X$ è una **superchiave**.

**Corollario 8.4.1.1.** *$R(T, F)$ con $F$ in copertura canonica è in BCNF se e solo se per ogni DF atomica non banale $X \to A \in F$, $X$ è una **superchiave** (ovvero è una chiave).*

Quindi un algoritmo per controllare se uno schema di relazione è in BCNF ha **complessità** $O(ap^2)$, dove $a$ è il numero di attributi in $T$ e $p$ è il numero di DF in $F$.

#### 8.4.1.1 Algoritmo di analisi

Questo algoritmo prevede che $R(T, F)$ venga decomposta in: $R_1(X, Y)$ e $R_2(X, Z)$ e su di esse si ripeta il procedimento.

---

**Input** $R(T, F)$ con $F$ copertura canonica
**Output** $\rho = \{R_1, R_2, R_m\}$ decomposizione in BCNF che preserva i dati
$\rho = \{R_1(T_1, F_1)\}$
**while** esiste in $\rho$ una $R_i(T_i, F_i)$ non in BCNF per la DF $X \to A$
**do**
  n = n + 1 # incrementa contatore relazioni
  $T_a = X^+$ # chiusura di $X$
  $F_a = \pi_{T_a}(F_i)$ # proiezione DF rispetto a $T_a$
  $T_b = T_i - X^+ + X$ # rimozione da attributi non in $X$ ma nella sua chiusura
  $F_b = \pi T_b(F_i)$ # proiezione DF rispetto a $T_b$
  $\rho = \rho - R_i + \{R_i < T_a, F_a >, R_n < T_b, F_b >\}$ # rimuove $R$ non BCNF per $X \to A$ e inserisce quelle corrette
**end**

---

Questo algoritmo **NON** garantisca la conservazione delle dipendenze ma preserva i dati.

**Esempio 8.4.1** (Applicazione dell'algoritmo)**.** Prendiamo

$$R(ABCDE, F = \{CE \to A, D \to E, CB \to E, CE \to B\})$$

e applichiamo l'algoritmo:

1. Consideriamo $F_1 = (CE \to A)$. $CE^+ = CEAB$, quindi $CE$ non è chiave (manca $D$). Decomponiamo in:
   - $R_1(CEAB)$: gli attributi di $CE^+$
   - $R_2(CED)$: l'attributo mancante $D$ e la chiave esterna $CE$
2. Proiettiamo le dipendenze funzionali
   - $R1(CEAB, \{CE \to A, CB \to E, CE \to B\})$
   - $R2(CED, \{D \to E\})$
3. Consideriamo:
   - $CE \to A$ e $CE \to B$, $CE^+ = CEAB$
   - $CB \to E$, $CB^+ = CEAB$

   quindi $R_1$ è in CBNF
4. Consideriamo $D \to E$, $D^+ = DE$ quindi $D$ non è chiave (manca $C$). Decomponiamo in:
   - $R_3(DE)$
   - $R_4(DC)$
5. La decomposizione ottenuta è

   $$\{R_1(CBEA), R_3(DE), R_4(DC)\}$$

   e preserva dati e dipendenze

### 8.4.2 Terza forma normale

**Definizione 8.4.2** (Terza forma normale)**.** *$R(T, F)$ è in terza forma normale se per ogni $X \to A \in F^+$, con $A \notin X$, $X$ è una superchiave o $A$ è primo.*

A differenza della BCNF, la terza forma normale **ammette** dipendenze **non banali** e da **non da chiave** se gli attributi a destra sono primi (possibili anomalie).

**Teorema 8.4.2.** $R(T, F)$ è in terza forma normale se per ogni $X \to A \in F$ non banale, allora $X$ è una superchiave, oppure $A$ è primo (ovvero è contenuto in almeno una chiave).

Ogni schema $R(T, F)$ ammette sempre una decomposizione che preserva i dati, preserva le dipendenze, ed è in 3NF. Tale decomposizione può essere ottenuta in **tempo polinomiale**.
Lo **svantaggio** sta nel fatto che, essendo la 3NF meno restrittiva della BCNF, accetta anche schemi che presentano delle **anomalie** e quindi certifica meno lo qualità dello schema ottenuto. In particolare tollera **ridondanze** sui dati.

#### 8.4.2.1 Algoritmo di sintesi

L'idea è che dato un insieme di attributi $T$ ed una copertura canonica $G$, si partiziona $G$ in gruppi $G_i$ tali che tutte le dipendenze in ognuno hanno la stessa parte sinistra. Quindi, da ogni $G_i$, si definisce uno schema di relazione composto da tutti gli attributi che vi appaiono, la cui chiave, detta **chiave sintetizzata**, è la parte sinistra comune.

```
Input    Un insieme R di attributi e un insieme F di dipendenze su R
Output   Una decomposizione ρ = {S_i}_{i=1...n} di R tale che preservi dati e dipendenze e ogni S_i sia in
     3NF, rispetto alle proiezioni di F su S_i

begin
  Passo 1. Trova una copertura canonica G di F e poni ρ = {}
  Passo 2. Sostituisci in G ogni insieme X → A_1, ..., X → A_h di dipendenze con lo stesso
       determinante, con la dipendenza X → A_1 ... A_h
  Passo 3. Per ogni dipendenza X...Y in G, metti uno schema con attributi XY in ρ
  Passo 4. Elimina ogni schema di ρ contenuto in un altro schema di ρ
  Passo 5. Se la decomposizione non contiene alcuno schema i cui attributi costituiscano una
       superchiave per R, aggiungi ad essa lo schema con attributi W, con W una chiave di R
end
```

## 8.5 Dipendenze multivalore

Esistono dipendenze di tipo non funzionale che di conseguenza non possono essere risolte dalla BCNF. Queste si presentano ogni volta che in una relazione si rappresentano **proprietà multivalore indipendenti**.
Le anomalie non dipendono solamente dal fatto che esista una proprietà multivalore ma dal fatto che questa stia con proprietà semplici o multivalore indipendenti.
Per evitare queste anomalie è stato necessario introdurre la **quarta forma normale** con relativo algoritmo.