

# **Basi di dati**

UniVR - Dipartimento di Informatica

**Fabio Irimie**

1° Semestre 2025/2026

# Indice

<b>1 Introduzione</b>	<b>3</b>
<b>2 Sistema informativo</b>	<b>3</b>
2.1 Base di dati . . . . .	3
2.1.1 Modello dei dati . . . . .	4
<b>3 Progettazione di una base di dati</b>	<b>5</b>
3.1 Progettazione dei dati . . . . .	5
3.2 Requisiti . . . . .	6
3.2.1 Progettazione concettuale . . . . .	6
3.2.2 Progettazione logica . . . . .	6
3.2.3 Progettazione fisica . . . . .	6
3.3 Strategie di progettazione . . . . .	6
3.3.1 Analisi di qualità dello shema . . . . .	7
<b>4 Progettazione Concettuale</b>	<b>8</b>
4.1 Entità . . . . .	8
4.1.1 Istanza . . . . .	8
4.2 Relazione . . . . .	9
4.2.1 Istanza . . . . .	9
4.2.2 Relazione ricorsiva . . . . .	10
4.3 Attributo . . . . .	10
4.3.1 Attributo opzionale e multivaleore . . . . .	11
4.3.2 Attributo composto . . . . .	11
4.4 Identificatore . . . . .	11
4.5 Cardinalità . . . . .	13
4.5.1 Valori possibili per MIN <sub>i</sub> . . . . .	13
4.5.2 Valori possibili per MAX <sub>i</sub> . . . . .	13
4.6 Generalizzazione di entità . . . . .	14
4.6.1 Proprietà delle istanze generalizzate . . . . .	15
4.6.2 Classificazione delle generalizzazioni . . . . .	15
4.6.3 Relazioni ternarie . . . . .	15
<b>5 Progettazione logica</b>	<b>16</b>
5.1 Modello relazionale . . . . .	16
5.2 Domini di base . . . . .	17
5.3 Relazione . . . . .	17
5.3.1 Accesso ai valori di una ennupla . . . . .	18
5.4 Progettazione nel modello relazionale . . . . .	19
5.4.1 Progettazione dei dati . . . . .	19
5.4.2 Relazione unica . . . . .	20
5.4.3 Anomalie . . . . .	20
5.4.4 Decomposizione della relazione unica . . . . .	21
5.4.5 Proprietà del modello relazionale . . . . .	24
5.4.6 Terminologia . . . . .	25
5.5 Valori nulli . . . . .	25
5.5.1 Osservazioni . . . . .	26
5.6 Vincoli di integrità . . . . .	26

5.6.1	Classificazione . . . . .	27
5.6.2	Vincoli di dominio . . . . .	27
5.6.3	Vincoli di tupla . . . . .	27
5.6.4	Vincoli di intrarelazionali . . . . .	28
5.6.5	Vincoli interrelazionali . . . . .	28
5.6.6	Notazione per indicare i vincoli strutturali . . . . .	29
5.7	Progettazione logica dallo schema concettuale . . . . .	30
5.7.1	Ristrutturazione dello schema concettuale . . . . .	30
5.7.2	Traduzione nel modello logico . . . . .	34
<b>6</b>	<b>Tecnologie NoSQL: sistemi document based</b>	<b>39</b>
6.1	Modello dei dati dei sistemi document-store . . . . .	40
6.1.1	Progettazione dei dati come documenti . . . . .	40
6.1.2	Rappresentazione di un documento . . . . .	42
6.2	Traduzione di uno schema ER in collezioni di documenti . . . . .	43
6.2.1	Regole di traduzione . . . . .	43
6.2.2	Etichettatura dello schema ER . . . . .	43
<b>7</b>	<b>Operazioni su una base di dati relazionale</b>	<b>57</b>
7.1	Algebra relazionale . . . . .	57
7.1.1	Operatori dell'algebra relazionale . . . . .	58
7.1.2	Operatori insiemistici . . . . .	58
7.1.3	Operatori specifici . . . . .	59
7.1.4	Operatori di giunzione . . . . .	61
7.2	Algebra con valori nulli . . . . .	64
7.2.1	Join esterni . . . . .	64
7.2.2	Ottimizzazione di espressioni DML . . . . .	64
7.2.3	Trasformazioni di equivalenza . . . . .	65
7.3	Calcolo relazionale . . . . .	66
7.3.1	Caratteristiche del linguaggio . . . . .	67
7.3.2	Sintassi del calcolo relazionale sulle tuple . . . . .	67
7.3.3	Semantica del calcolo relazionale sulle tuple . . . . .	68
7.3.4	Semantica di una interrogazione . . . . .	69
7.3.5	Operatori insiemistici nel calcolo relazionale sulle tuple . . . . .	70
7.3.6	Semantica dei quantificatori . . . . .	70
7.4	Interrogazioni nei sistemi document based . . . . .	71
7.4.1	Operatori logici e di confronto in MongoDB . . . . .	73
7.4.2	Interrogazione di dati encapsulati . . . . .	74
7.4.3	Interrogazioni con proiezione . . . . .	74
7.4.4	Interrogazioni con join . . . . .	75
7.4.5	Interrogazioni su dati encapsulati con condizioni multiple . . . . .	76

# 1 Introduzione

Le basi di dati sono raccolte di dati strutturati, organizzati in modo tale da permettere un facile accesso. Questi dati sono persistenti, ovvero rimangono memorizzati anche dopo la chiusura del programma che li ha creati.

## 2 Sistema informativo

Un sistema informativo è l'insieme delle attività umane e dei dispositivi di memorizzazione ed elaborazione che organizza e gestisce l'informazione di interesse per un'organizzazione di dimensioni qualsiasi. Non contiene necessariamente dati memorizzati in un computer.

Un sistema informativo è composto da:

- **Dato:** è l'elemento di conoscenza di base costituito da simboli che devono essere elaborati
- **Informazione:** è l'interpretazione dei dati che permette di ottenere una conoscenza più o meno esatta di fatti e situazioni

### 2.1 Base di dati

**Definizione 2.1.** Una **base di dati** è una **collezione di dati persistenti** utilizzati per rappresentare **con tecnologia informatica** le informazioni di interesse per un **sistema informativo**

La soluzione convenzionale per la gestione dei dati è l'uso di file, ma questa presenta alcuni problemi:

- Scarsa efficienza nell'accesso ai dati (accesso sequenziale)
- Ridondanza nei dati
- Inconsistenza nei dati (aggiornamenti parziali)
- Progettazione dei dati replicata per ogni applicazione

Per risolvere questi problemi si è creato un livello di astrazione maggiore tra le applicazioni e il filesystem, ovvero il **Data Base Management System (DBMS)**.

**Definizione 2.2.** Un **DBMS** è un sistema che gestisce su **memoria secondaria** collezioni di dati (chiamate "basi di dati"). Le caratteristiche principali sono:

- Grandi
- Condivise, cioè accessibili da più utenti
- Persistenti

Un DBMS assicura:

- Affidabilità, cioè nessuna perdita di dati
- Privatezza
- Accesso efficiente

### 2.1.1 Modello dei dati

Un **modello dei dati** è un insieme di strutture che permettono di descrivere una base di dati. Per accedere a questi dati si usano delle **interrogazioni**, cioè delle richieste, in un linguaggio dichiarativo specifico, che permettono di ottenere i dati desiderati.

Ci sono diversi linguaggi per interagire con un DBMS:

- Linguaggio per la definizione dei dati (DDL), consente di definire la struttura della base di dati
- Linguaggio per l'interrogazione e aggiornamento dei dati (DML), consente di interrogare e aggiornare i dati
  - Linguaggio di interrogazione: estrae informazioni da una base di dati, ad esempio SQL, algèbre relazionale, calcolo relazionale
  - Linguaggio di manipolazione: popola la base di dati, modifica il suo contenuto con aggiunge, cancellazioni e variazioni sui dati, ad esempio SQL

Il modello di dati è un insieme di **costrutti** forniti dal DBMS per descrivere la struttura e le proprietà dell'informazione contenute in una base di dati.

Ci sono diversi tipi di modelli di dati:

- **Modelli di dati del passato:**
  - Modello reticolare
  - modello gerarchico
- **Modelli di dati attuali:**
  - Modello relazionale
  - Modello ad oggetti
  - Modello a oggetti-relazionale
  - Modello basato su documenti (JSON)
  - Modelli NoSQL

I modelli vengono utilizzati per creare:

- **Schema di una base di dati:** è la descrizione della struttura e delle proprietà di una specifica base di dati fatta utilizzando i costrutti del modello dei dati (lo schema di una base di dati è invariante nel tempo)
- **Istanza di una base di dati:** è costituita dai **valori effettivi** che in un certo istante popolano le strutture dati (l'istanza di una base di dati varia nel tempo)

Lo schema di una base di dati è diviso in tre livelli:

- **Schema esterno**: è la visione dell'utente della base di dati, cioè la parte di base di dati che interessa a un particolare utente o gruppo di utenti
- **Schema logico**: è la visione globale della base di dati, cioè la struttura logica della base di dati che descrive tutti i dati e le relazioni tra essi
- **Schema interno**: è la rappresentazione fisica della base di dati, cioè il modo in cui i dati sono effettivamente memorizzati nella memoria secondaria

Le proprietà dello schema sono:

- **Indipendenza fisica**: lo schema logico della base di dati è completamente indipendente dallo schema interno
- **Indipendenza logica**: gli schemi esterni della base di dati sono indipendenti dallo schema logico

### 3 Progettazione di una base di dati

Il ciclo di vita di un processo di automazione di un sistema informativo è diviso in diverse fasi:

- **Studio di fattibilità**: si valuta se l'automazione del sistema informativo è possibile e conveniente
- **Raccolta e analisi dei requisiti**: si individuano proprietà e funzionalità del sistema (dati e applicazioni) producendo una descrizione completa ma informale
- **Progettazione**: si produce una descrizione formale del sistema informativo

La progettazione si divide in due parti principali che vanno di pari passo:

- **Progettazione dei dati**: si produce una descrizione formale dei dati (schema). Una volta progettati i dati vengono implementati in un DBMS
- **Progettazione delle applicazioni**: si produce una descrizione formale delle applicazioni (specifiche)

Una volta implementati i dati e le applicazioni si passa alla fase di **validazione e collaudo**

#### 3.1 Progettazione dei dati

Una metodologia di progettazione dei dati è costituita da:

- **Decomposizione**: dividere in passi le attività di progetto
- **Strategie**: individuare un insieme di strategie e criteri di scelta da seguire
- **Modelli di riferimento**: utilizzare modelli di dati e tecniche di progettazione consolidate

Una buona metodologia deve essere:

- Generale
- Facile da usare
- Deve produrre un risultato di qualità

### 3.2 Requisiti

#### 3.2.1 Progettazione concettuale

La progettazione concettuale è la prima fase della progettazione dei dati. Lo scopo è quello di produrre una descrizione formale dei dati (schema concettuale). Lo schema deve essere **indipendente dall'implementazione**.

Non è solo un progetto intermedio, ma costituisce anche una porzione del risultato finale perché rappresenta una descrizione di **alto livello** del contenuto della base di dati, comprensibile anche per utenti poco esperti.

#### 3.2.2 Progettazione logica

La progettazione logica è la seconda fase della progettazione dei dati. Lo scopo è quello di tradurre lo schema concettuale in uno schema logico in modo da poterlo utilizzare su un sistema specifico. Lo schema logico infatti è dipendente dalle tecnologie utilizzate. Bisogna tenere anche in considerazione le operazioni più frequenti che le applicazioni effettueranno sulla base di dati.

#### 3.2.3 Progettazione fisica

La progettazione fisica è la terza fase della progettazione dei dati. L'obiettivo è quello di ottimizzare l'accesso ai dati completando lo schema logico con i parametri relativi alla memorizzazione fisica dei dati e con gli opportuni metodi d'accesso (**indici**).

### 3.3 Strategie di progettazione

Lo sviluppo di uno schema concettuale può essere visto come un processo di ingegnerizzazione ed è quindi possibile applicare anche a tale processo le strategie classiche:

- **Top-Down:** consiste nel considerare le specifiche **globalmente** e produrre uno schema iniziale completo ma con **pochi concetti** molto astratti. Si va poi a **raffinare** i concetti astratti fino ad arrivare allo schema concettuale completo in ogni dettaglio
- **Bottom-Up:** consiste nel **decomporre** le specifiche iniziali in parti elementari e produrre uno schema iniziale dettagliato
- **Inside-Out:** consiste nell'individuare nelle specifiche alcuni concetti importanti (detti concetti guida) e partendo da quelli generare gli schemi per i relativi concetti. Si procede fondendo gli schemi precedenti per generare lo schema finale.

### 3.3.1 Analisi di qualità dello shema

L'analisi della qualità dello schema concettuale prodotto può essere suddivisa in diverse fasi:

- **Verifica della correttezza:** Uno schema concettuale è corretto se utilizza correttamente i costrutti del modello concettuale adottato (nel nostro caso E-R). I possibili errori sono:

- Errori sintattici: si verificano quando i costrutti vengono usati senza seguire le regole sintattiche
- Errori semantici: si verificano quando i costrutti vengono usati senza seguire le regole semantiche, cioè la loro definizione

- **Verifica della completezza:** Uno schema concettuale è completo se rappresenta tutti i requisiti espressi nelle specifiche.

- **Verifica di minimalità:** Uno schema è minimale quando tutti i concetti descritti nei requisiti sono rappresentati nello schema **una volta sola**. Uno schema non minimale contiene delle **ridondanze**. I casi di probabile ridondanza sono:

- Ereditarietà nelle relazioni: si manifesta quando due entità padri sono in relazione e le loro entità figlie sono anch'esse in relazione. Per far sì che non ci sia ridondanza in questo caso bisogna verificare che le due relazioni non rappresentino lo stesso concetto
- Cicli di relazioni: si manifesta quando esistono dei cicli tra le relazioni di più entità. Non è detto che ci sia sicuramente ridondanza, ma per evitarla bisogna verificare se la relazione  $R_i$  si possa ottenere dalla composizione delle altre relazioni. Se è così allora  $R_i$  va eliminata in quanto ridondante.

- **Verifica di leggibilità:** Uno schema concettuale è leggibile quando rappresenta tutti i requisiti in modo naturale e facilmente comprensibile. Alcuni consigli per migliorare la leggibilità sono:

- Curare la scelta dei nomi delle componenti dello schema
- Disegnare bene lo schema

Si può migliorare la leggibilità anche semplificando lo schema. Le relazioni ternarie non sono facili da leggere e interpretare, quindi solitamente la loro eliminazione può comportare una migliore leggibilità dello schema. Le operazioni possibili sono:

- Trasformare una relazione ternaria in un'entità (sempre applicabile)
- Riduzione di una relazione ternaria a due relazioni binarie  
(**precondizione:** una delle entità partecipa alla relazione ternaria con cardinalità (1,1) (spiegata nel capitolo 4.5))
- Trasformare una relazione sovrapposta in una generalizzazione esclusiva esplicitando l'entità che rappresenta la sovrapposizione

## 4 Progettazione Concettuale

È un modello, formale e non ambiguo, utilizzato per la progettazione concettuale di una base di dati. Fornisce strumenti formali (costrutti), con sintassi grafica, per specificare la struttura e le proprietà dei dati da rappresentare indipendentemente dalla tecnologia.

Ogni costrutto viene definito specificando:

- Il suo significato (o semantica)
- La sua sintassi grafica
- La rappresentazione delle sue istanze (o occorrenze)

Progettare indipendentemente dalle tecnologie significa:

- **Non considerare** eventuali ottimizzazioni
- **Considerare** tutti i requisiti senza semplificazioni o convenzioni
- **Considerare** sempre i processi di generazione e modifica dei dati per verificare che ogni situazione sia rappresentabile da un'istanza "pulita" della base di dati

### 4.1 Entità

Un'entità E rappresenta una **classe di oggetti** che hanno le seguenti caratteristiche:

- **Proprietà comuni**
- **Esempio autonoma** rispetto ad altre classi di oggetti
- **Identificazione univoca**, cioè esiste una chiara corrispondenza tra gli oggetti istanze di entità e concetti istanziati nel sistema informativo

Un entità si rappresenta con un rettangolo che contiene il nome dell'entità:



Figura 1: Rappresentazione grafica di un'entità

#### 4.1.1 Istanza

Un'istanza dell'entità E è un **oggetto** appartenente alla classe rappresentata da E. Si indica con  $I(E)$  l'insieme delle istanze di E che esistono nella base di dati in un certo istante e alla creazione della base di dati è vuota:  $I(E) = \emptyset$ .

**Esempio 4.1.** Rappresentiamo con il costrutto entità il concetto di **persona**. Bisogna gestire nella base di dati le informazioni che descrivono un gruppo di persone.



Persona

Figura 2: Rappresentazione grafica dell'entità Persona

L'insieme delle istanze dell'entità Persona è il seguente:

$$I(\text{Persona}) = \{p_1, p_2, p_3, \dots\}$$

## 4.2 Relazione

Una relazione  $R$  rappresenta un **legame logico** tra **due o più entità**. Può esserci anche una relazione all'entità stessa (relazione ricorsiva).

Una relazione si rappresenta nello schema con un rombo a cui si collegano attraverso delle linee le entità coinvolte nella relazione. Il nome della relazione viene scritto a fianco al rombo:



Figura 3: Rappresentazione grafica di una relazione tra due entità

### 4.2.1 Istanza

Data una relazione  $R$  tra  $n$  entità  $E_1, \dots, E_n$  un'istanza della relazione  $R$  è una **ennupla di istanze di entità**:

$$(e_1, \dots, e_n) \text{ dove } e_i \in I(E_i) \text{ per } 1 \leq i \leq n$$

La popolazione di  $R$  rappresenta l'insieme delle coppie di istanze delle entità  $E$  e  $F$  che sono in relazione in un certo istante:

$$I(R) = \{(e_i, f_j) \mid e_i \in I(E), f_j \in I(F)\}$$

**Esempio 4.2.** Supponiamo che nello schema ci siano le entità **Persona** e **Comune**, bisogna gestire la **Residenza** delle persone nei comuni italiani.

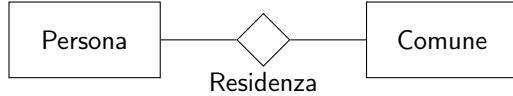


Figura 4: Rappresentazione grafica della relazione Residenza

Ciò implica che per esistere un'istanza di residenza devono esistere un'istanza

di persona e un'istanza di comune.

Data una relazione R tra n entità  $\{E_1, E_2, \dots, E_n\}$  vale **sempre** la seguente proprietà sull'insieme delle istanze  $I(R)$ :

$$I(R) \subseteq I(E_1) \times I(E_2) \times \dots \times I(E_n)$$

La conseguenza di questa proprietà è che non è possibile rappresentare la stessa ennupla più volte.

#### 4.2.2 Relazione ricorsiva

È una relazione binaria sulla stessa entità:

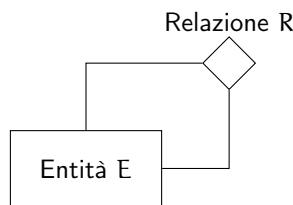


Figura 5: Rappresentazione grafica di una relazione ricorsiva

### 4.3 Attributo

Rappresenta una proprietà elementare di un'entità o di una relazione. Ogni attributo di un'entità o di una relazione associa ad ogni istanza **un solo** valore appartenente ad un dominio di valori ammissibili. Può essere visto come una funzione che ha come dominio le istanze dell'entità (o relazione) e come codominio l'insieme dei valori ammissibili:

$$f_A : I(E) \rightarrow D$$

dove a è un attributo dell'entità E, mentre  $I(E)$  l'insieme delle istanze di E e D è l'insieme dei valori ammissibili.

La sintassi grafica di un attributo è un cerchio **vuoto** collegato con una linea all'entità con accanto il nome dell'attributo:

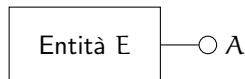


Figura 6: Rappresentazione grafica di un attributo di un'entità



Figura 7: Rappresentazione grafica di un attributo di una relazione

**Esempio 4.3.** Rappresentiamo il concetto di persona tramite un'entità, bisogna gestire nella base di dati il nome e il cognome di un gruppo di persone.

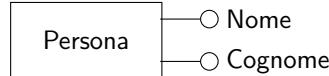


Figura 8: Rappresentazione grafica dell'entità Persona con gli attributi Nome e Cognome

#### 4.3.1 Attributo opzionale e multivaleore

L'attributo opzionale o multivaleore si ottiene da un attributo normale specificando un vincolo di cardinalità (spiegato più avanti per le relazioni 4.5) sui valori che l'attributo può assumere (il default è (1,1)). I valori possibili sono i seguenti:

- (0,1): attributo opzionale
- (1,N): attributo multivaleore obbligatorio
- (0,N): attributo multivaleore opzionale

La sintassi grafica per rappresentare un attributo opzionale o multivaleore è la seguente:

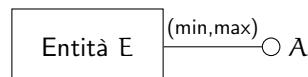


Figura 9: Rappresentazione grafica di un attributo opzionale o multivaleore

#### 4.3.2 Attributo composto

Permette di raggruppare gli attributi di un'entità o di una relazione che hanno un significato comune.

La sintassi grafica per rappresentare un attributo composto è la seguente:

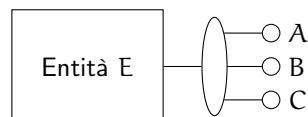


Figura 10: Rappresentazione grafica di un attributo composto

### 4.4 Identificatore

Data un'entità E, un identificatore è un insieme di proprietà (attributi e/o relazioni) che **identificano univocamente** ogni istanza di E. Un insieme di proprietà identifica univocamente le istanze di un'entità E se **non esistono** due istanze di E che presentano gli stessi valori o istanze nelle proprietà dell'insieme.

La sintassi grafica di un identificatore per un'entità con un solo attributo  $a$  è la seguente:

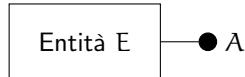


Figura 11: Rappresentazione grafica di un identificatore con un solo attributo

Per un identificatore costituito da più attributi  $a, b$  la sintassi grafica è la seguente:



Figura 12: Rappresentazione grafica di un identificatore con più attributi

Può anche esistere un identificatore costituito da una relazione  $R$  (deve essere una funzione):

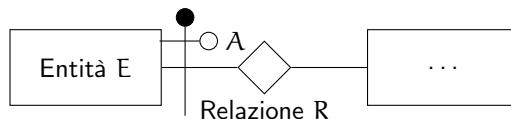


Figura 13: Rappresentazione grafica di un identificatore costituito da un attributo e una relazione

Oppure un identificatore può essere costituito soltanto da una relazione  $R$ :



Figura 14: Rappresentazione grafica di un identificatore costituito da una relazione

Ci sono due tipi di identifieri:

- **Identifieri interni:** sono costituiti solo da attributi dell'entità
- **Identifieri esterni:** sono costituiti da almeno una relazione con un'altra entità

**Attenzione:** Non esistono identifieri sulle relazioni perché ogni relazione è già univocamente identificata dalle istanze delle entità coinvolte.

**Esempio 4.4.** Rappresentiamo il concetto di persona tramite un'entità, bisogna gestire nella base di dati il codice fiscale, il nome, il cognome e la data di nascita di un gruppo di persone

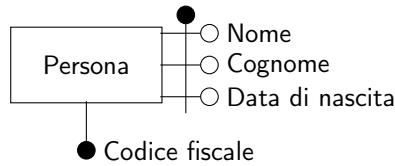


Figura 15: Esempio di identificatore interno

**Definizione 4.1.** Un **vincolo di identificazione** limita la popolazione di un'entità impedendo l'esistenza di due istanze con gli stessi valori nelle proprietà che costituiscono l'identificatore.

La scelta dell'identificatore va sempre fatta considerando le proprietà significative per il sistema informativo. A **livello concettuale** è quindi da **evitare** l'introduzione di nuovi attributi identificatori (ad esempio l'ID).

## 4.5 Cardinalità

Tra due relazioni esistono dei vincoli di cardinalità che limitano il numero di istanze di una entità che possono essere in relazione con una singola istanza dell'altra entità. Data una relazione R i vincoli di cardinalità vengono specificati per ogni entità  $E_i$  coinvolta nella relazione R e specificano il numero **minimo** e **massimo** di istanze di R a cui un'istanza di  $E_j$  deve o può partecipare.

La sintassi grafica per rappresentare i vincoli di cardinalità è la seguente:

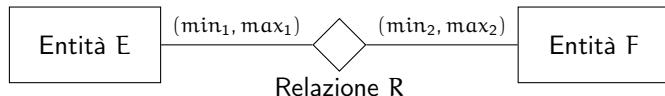


Figura 16: Rappresentazione grafica dei vincoli di cardinalità

### 4.5.1 Valori possibili per $\text{MIN}_i$

I possibili valori che il minimo  $\text{MIN}_i$  può assumere sono:

- 0: Indica che la partecipazione alla relazione R delle istanze di  $E_i$  è **opzionale**
- 1: Indica che la partecipazione alla relazione R delle istanze di  $E_i$  è **obbligatoria**
- $\text{num} > 1$ : Indica che per ogni istanza di  $E_i$  devono essere presenti almeno  $\text{num}$  occorrenze della relazione R che la coinvolgono

### 4.5.2 Valori possibili per $\text{MAX}_i$

- 1: Indica che un'istanza di  $E_i$  può **al massimo** partecipare a una sola occorrenza della relazione R (se R è binaria questo indica che R è **una funzione**)

- N: Indica che un'istanza di  $E_i$  può partecipare a più occorrenze della relazione R senza limite massimo
- num > 1: Indica che per ogni istanza di  $E_i$  possono essere presenti **al massimo** num occorrenze della relazione R che la coinvolgono

**Esempio 4.5.** Rappresentiamo il concetto di persona e comune tramite due entità, bisogna gestire la residenza delle persone nei comuni italiani. Questi requisiti non rappresentano la realtà perché una persona deve avere obbligatoriamente una residenza e può avere al massimo una residenza, mentre un comune può avere zero o più persone residenti.

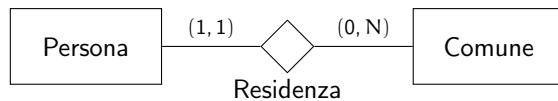


Figura 17: Esempio di Residenza con i vincoli di cardinalità

**Definizione 4.2.** Una relazione R può partecipare ad un identificatore esterno di un'entità E **solo se** tale entità partecipa alla relazione R con vincolo di cardinalità (1, 1), quindi se R è una **funzione** che associa ad ogni istanza di E una e una sola istanza dell'altra entità coinvolta in R. Questo vale anche quando R è la sola proprietà che partecipa all'identificatore.

Si potrebbe utilizzare anche il diagramma UML per rappresentare i concetti del modello Entità-Relazione, ma questa rappresentazione segue regole diverse, ad esempio la posizione dove specificare nel diagramma UML i vincoli di cardinalità è invertita rispetto all'ER.

## 4.6 Generalizzazione di entità

La generalizzazione è un legame logico (simile ad un'ereditarietà tra classi) tra un'entità padre E e n ( $n > 0$ ) entità figlie  $E_1, E_2, \dots, E_n$  dove E rappresenta una classe di oggetti più generale rispetto alle classi di oggetti rappresentate dalle entità figlie.

La sintassi grafica per rappresentare una generalizzazione è la seguente:

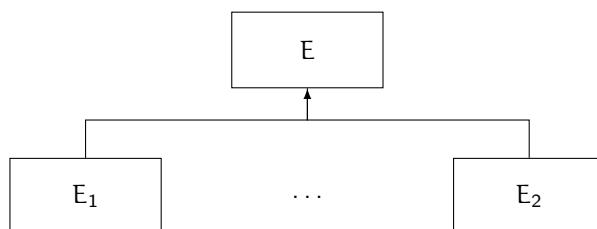


Figura 18: Esempio di Residenza con i vincoli di cardinalità

#### 4.6.1 Proprietà delle istanze generalizzate

- Ogni istanza di un'entità figlia  $E_i$  è anche istanza dell'entità padre  $E$
- Ogni proprietà (attributi, identificatori e relazioni) dell'entità padre  $E$  è anche proprietà di ogni entità figlia  $E_i$

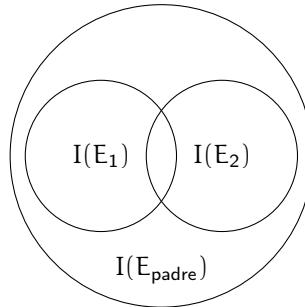


Figura 19: Istanze di entità generalizzate

#### 4.6.2 Classificazione delle generalizzazioni

Una generalizzazione può essere classificata secondo due criteri:

- **Totale** se ogni istanza dell'entità padre  $E$  è anche istanza di **almeno una** delle entità figlie  $E_i$ , altrimenti si dice **parziale**
- **Esclusiva** se ogni istanza dell'entità padre  $E$  è anche istanza di **al massimo una** delle entità figlie  $E_i$ , altrimenti si dice **sovrapposta**

Per indicare la classificazione di una generalizzazione si mettono le seguenti notazioni accanto alla freccia della generalizzazione:

- (p,s): parziale e sovrapposta:
- (p,e): parziale ed esclusiva: non si possono avere istanze comuni tra le entità figlie
- (t,s): totale e sovrapposta: non si possono avere istanze dell'entità padre che non siano istanze di almeno una delle entità figlie
- (t,e): totale ed esclusiva: ogni istanza dell'entità padre è istanza di **esattamente una** delle entità figlie

#### 4.6.3 Relazioni ternarie

Una relazione ternaria è una relazione che coinvolge tre entità e si usa quando:

- Un'istanza del concetto per esistere nel sistema informativo richiede sempre la presenza di tre istanze di entità (una per ogni entità coinvolta nella relazione)
- Non esistono eccezioni al punto precedente

- Non esistono situazioni nelle quali la terna di entità debba essere rappresentata più volte

La sintassi grafica per rappresentare una relazione ternaria è la seguente (i vincoli di cardinalità sono richiesti, ma qua sono omessi):

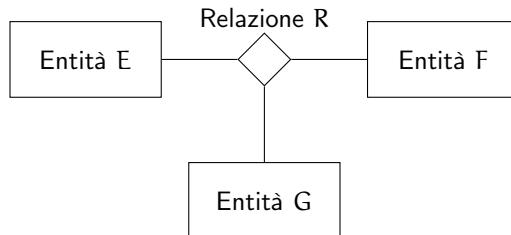


Figura 20: Rappresentazione grafica di una relazione tra due entità

Un uso errato avviene quando per rappresentare lo stato del sistema informativo sarebbe necessario che un'istanza della relazione **contenga solo una coppia e non una terna** di istanze di entità (cioè quando manca un attore). Quindi in questo caso si può sostituire la relazione ternaria con 3 relazioni binarie indipendenti.

L'utilizzo delle relazioni ternarie è utile per "storicizzare" le relazioni binarie. Nella rappresentazione di una relazione binaria storizzata è necessario poter rappresentare più volte **la stessa coppia in tempi diversi**. Pertanto la relazione binaria può essere trasformata in una relazione ternaria includendo l'entità **Tempo** come nuovo attore nella relazione in modo da poter rappresentare più volte la stessa coppia in istanti di tempo diversi.

## 5 Progettazione logica

### 5.1 Modello relazionale

Il **modello relazionale** è un modello dei dati, quindi permette di definire le proprietà che la popolazione di una base di dati deve rispettare. I costrutti principali del modello relazionale sono:

- **Domini di base**: insieme di valori atomici (non scomponibili) che rappresentano i tipi di dati elementari
- **Relazione (o tabella)**: rappresenta un insieme di entità o di relazioni tra entità. Sono definite come un insieme di ennuple o tuple.
- **Superchiavi, chiavi e chiavi primarie**: insieme di attributi che identificano univocamente ogni ennupla di una relazione
- **Vincoli di integrità referenziale**: vincoli che garantiscono la coerenza tra i dati memorizzati in relazioni diverse
- **Vincoli di integrità generici**: vincoli che garantiscono la coerenza dei dati memorizzati in una singola relazione

## 5.2 Domini di base

Sono i domini da cui si scelgono i valori delle proprietà delle istanze di informazione da rappresentare. I domini tipici sono:

- Caratteri
- Stringhe di caratteri
- Numeri interi
- Numeri decimali a virgola fissa
- Numeri decimali a virgola mobile
- Domini del tempo, per rappresentare istanti e intervalli di tempo
- ecc...

I domini presenti variano a seconda del sistema, ma in questo corso vedremo il linguaggio SQL nel sistema PostgreSQL.

## 5.3 Relazione

Una relazione può essere vista come una tabella.

**Esempio 5.1.** Ad esempio:

Milano	20100	1.300.000
Verona	37100	350.000
Brescia	25100	250.000

Tabella 1: Esempio di tabella che rappresenta alcune città italiane

Una tabella è un contenitore di dati la cui struttura è caratterizzata da una lista di colonne:

- I dati sono scritti nelle righe dove **ogni riga** descrive le caratteristiche di **un'istanza dell'informazione da rappresentare**
- I valori contenuti nell' **colonne** descrivono sempre la **stessa proprietà delle istanze** di informazione da rappresentare

**Definizione 5.1** (Definizione di relazione come insieme di ennuple (Lists)). Dati  $n$  insiemi di valori (domini)  $D_1, \dots, D_n$ , con  $n > 0$  e indicato con  $D_1 \times \dots \times D_n$  il loro prodotto cartesiano:

$$D_1 \times \dots \times D_n = \{(v_1, \dots, v_n) \mid v_1 \in D_1 \wedge \dots \wedge v_n \in D_n\}$$

una relazione  $\rho$  di grado  $n$  è un qualsiasi **sottoinsieme** di  $D_1 \times \dots \times D_n$ :

$$\rho \subseteq D_1 \times \dots \times D_n$$

dove  $(v_1, \dots, v_n)$  è una ennupla della relazione e  $|\rho|$  è la cardinalità della relazione (numero di ennupla).

**Nota:**

- I domini  $D_1, \dots, D_n$  possono essere a **cardinalità infinita**, mentre le relazioni sono sempre a **cardinalità finita**
- Dalla definizione si deduce inoltre che:
  - **Non è definito alcun ordinamento** sulle ennupla di una relazione
  - **Non sono ammessi duplicati** di una ennupla
  - Nella definizione di relazione come insieme di ennupla, i valori delle ennupla sono ordinati

**Esempio 5.2.** Riprendendo l'esempio 5.1, la relazione può essere vista come insieme di ennupla (relazione città):

$$\rho = \left\{ (\text{Milano}, 20100, 1300000), (\text{Verona}, 37100, 350000), (\text{Brescia}, 25100, 250000) \right\}$$

dove:

$$\rho \subseteq D_1 \times D_2 \times D_3$$

### 5.3.1 Accesso ai valori di una ennupla

Se  $t$  è una ennupla  $(v_1, \dots, v_n)$  il valore posto in  $i$ -esima posizione si indica con la notazione:

$$t[i]$$

Questa modalità di accesso ai valori però non è efficace per l'uso pratico delle relazioni. Si preferisce quindi assegnare un **nome alle colonne**; ciò conduce all'introduzione della definizione di relazione come insieme di **tuple**.

**Definizione 5.2** (Definizione di relazione come insieme di tuple (Mappings)). Sia  $X$  un insieme di nomi e sia  $\Delta$  l'insieme di tutti i domini di base ammessi dal modello. Si definisce la funzione:

$$\text{DOM} : X \rightarrow \Delta$$

Tale funzione associa ad ogni nome  $A \in X$  un dominio  $\text{DOM}(A) \in \Delta$ . I nomi di  $X$  si definiscono **attributi**. Una tupla  $t$  su  $X$  è una funzione:

$$t : X \rightarrow \bigcup_{A \in X} \text{DOM}(A)$$

dove:

$$t[A] = v \in \text{DOM}(A)$$

Quindi una relazione su X è un **insieme di tuple su X**, dove X è l'insieme di attributi della relazione.

**Nota:**

- Una relazione è un **insieme** di tuple e quindi **non può contenere tuple duplicate**
- I domini per gli attributi possono essere solo domini di base, **non sono ammessi altri domini, né il prodotto cartesiano di domini**. Essenzialmente non sono ammessi attributi anidati
- In generale una base di dati relazionale è composta da **più relazioni**

**Esempio 5.3.** Consideriamo l'esempio 5.1. Definiamo l'insieme di attributi:

$$X = \{\text{Nome}, \text{CAP}, \text{Abitanti}\}$$

Definiamo il dominio di ciascun attributo:

$$\text{DOM}(\text{Nome}) = \text{Stringhe di caratteri}$$

$$\text{DOM}(\text{CAP}) = \text{Numeri interi}$$

$$\text{DOM}(\text{Abitanti}) = \text{Numeri interi}$$

La tabella della relazione è rappresentata come un insieme di tuple:

$$\rho_X = \{t_1, t_2, t_3\}$$

dove:

$$t_1[\text{Nome}] = \text{Milano} \quad t_1[\text{CAP}] = 20100 \quad t_1[\text{Abitanti}] = 1300000$$

$$t_2[\text{Nome}] = \text{Verona} \quad t_2[\text{CAP}] = 37100 \quad t_2[\text{Abitanti}] = 350000$$

$$t_3[\text{Nome}] = \text{Brescia} \quad t_3[\text{CAP}] = 25100 \quad t_3[\text{Abitanti}] = 250000$$

## 5.4 Progettazione nel modello relazionale

### 5.4.1 Progettazione dei dati

Nel modello relazionale lo schema è costituito da un insieme di relazioni (o tabelle). Le dipendenze tra proprietà elementari (attributi) sono costituite dalle **dipendenze funzionali** che in questo corso sono sostituite dalla progettazione concettuale. Consideriamo i seguenti requisiti da implementare nel modello relazionale:

Vogliamo rappresentare in una base di dati relazionale le informazioni sulla proprietà degli appartamenti siti nel comune di Verona.

- Per ogni appartamento vogliamo memorizzare: il codice ecografico (univoco), la categoria catastale e l'indirizzo composto da: via, numero civico, subalterno.

- Per ogni proprietario si registra: il codice fiscale, il nome, il cognome e la data di nascita.
- Un appartamento può essere in comproprietà e un proprietario può possedere più appartamenti

#### 5.4.2 Relazione unica

Una soluzione possibile è rappresentata da un'unica relazione che contiene tutte le informazioni sugli appartamenti e sui proprietari:

Categoria	Via	Civico	Sub	Codice ECO	Codice Fiscale	Nome	Cognome	Data Nas
A1	Roma	23	A	RMAXXX	RSSMRR75D	Mario	Rossi	1/1/1975
A1	Roma	23	A	RMAXXX	BNCMRA80F	Maria	Bianchi	10/9/1980
A3	Milano	9		MILYYY	BNCMRA80F	Maria	Bianchi	10/9/1980
A3	Milano	9		MILYYY	BNCPLO77A	Paolo	Bianchi	3/1/1977
A1	Torino	2	C	TORZZZ	BNCPLO77A	Paolo	Bianchi	3/1/1977
A2	Garibaldi	33		GARWWW	RSSMRR75D	Mario	Rossi	1/1/1975

Figura 21: Schema relazionale con una sola relazione Proprietà

Questa tabella associa ogni appartamento ad un proprietario perché i dati dell'appartamento e della persona fanno parte della stessa riga. Il problema è che non c'è distinzione tra queste due categorie di informazioni. Questo porta a diversi problemi:

1. **Mancata separazione delle informazioni:** le due categorie di informazione non possono esistere separatamente.
2. **Ridondanza dei dati:** se un proprietario possiede più appartamenti i suoi dati vengono memorizzati più volte. Questo comporta un aumento dello spazio di memorizzazione e una possibile inconsistenza dello stesso dato.

#### 5.4.3 Anomalie

Le problematiche descritte nel paragrafo precedente portano sono descritte in modo più dettagliato come **anomalie** che si dividono in:

- **Anomalia di aggiornamento:** per aggiornare il valore di un attributo si è obbligati a modificare tale valore su più tuple
- **Anomalia di inserimento:** per inserire una nuova tupla è necessario inserire valori al momento sconosciuti (sostituibili da valori nulli) per gli attributi non disponibili
- **Anomalia di cancellazione:** per cancellare una tupla è necessario cancellare valori ancola validi oppure inserire valori nulli per gli attributi da cancellare

#### 5.4.4 Decomposizione della relazione unica

Riprendiamo l'esempio precedente con una singola relazione e decomponiamola separando le diverse categorie di informazioni in relazioni distinte. Ci sono diversi modi per effettuare questa decomposizione:

1. Separazione in due relazioni distinte per Appartamento e Proprietario

APPARTAMENTO					PROPRIETARIO			
Categoria	Via	Civico	Sub	Codice ECO	Codice Fiscale	Nome	Cognome	Data Nas
A1	Roma	23	A	RMAXXX	RSSMRR75D	Mario	Rossi	1/1/1975
A3	Milano	9		MILYYY	BNCMRA80F	Maria	Bianchi	10/9/1980
A1	Torino	2	C	TORZZZ	BNCPLO77A	Paolo	Bianchi	3/1/1977
A2	Garibaldi	33		GARWWW				

Figura 22: Schema relazionale con relazioni distinte per Appartamento e Proprietario

Questa separazione risolve i problemi di ridondanza e anomalie di aggiornamento. Però manca l'informazione che mette in relazione la persona con l'appartamento. Quindi siccome si è persa un'informazione, questa decomposizione è detta **decomposizione con perdita**.

2. Separazione in due relazioni distinte per Appartamento e Proprietario con l'aggiunta di un'informazione identificante per la relazione Appartamento, in questo caso è

- codice fiscale della Persona nella relazione Appartamento

APPARTAMENTO					
Categoria	Via	Civico	Sub	Codice ECO	Codice Fiscale
A1	Roma	23	A	RMAXXX	RSSMRR75D
A1	Roma	23	A	RMAXXX	BNCMRA80F
A3	Milano	9		MILYYY	BNCMRA80F
A3	Milano	9		MILYYY	BNCPLO77A
A1	Torino	2	C	TORZZZ	BNCPLO77A
A2	Garibaldi	33		GARWWW	RSSMRR75D

Figura 23: Relazione Appartamento con codice fiscale

PROPRIETARIO			
Codice Fiscale	Nome	Cognome	Data Nas
RSSMRR75D	Mario	Rossi	1/1/1975
BNCMRA80F	Maria	Bianchi	10/9/1980
BNCPLO77A	Paolo	Bianchi	3/1/1977

Figura 24: Relazione Proprietario con codice ecografico

Siccome non esistono i puntatori bisogna duplicare i valori identificanti nelle due relazioni. Anche questa decomposizione ha dati ridondanti.

3. Separazione in due relazioni distinte per Appartamento e Proprietario con l'aggiunta di un informazione identificante per la relazione Proprietario, in questo caso è:

- codice ecografico dell'Appartamento nella relazione Proprietario

APPARTAMENTO					
Categoria	Via	Civico	Sub	Codice ECO	Codice Fiscale
A1	Roma	23	A	RMAXXX	RSSMRR75D
A1	Roma	23	A	RMAXXX	BNCMRA80F
A3	Milano	9		MILYYY	BNCMRA80F
A3	Milano	9		MILYYY	BNCPLO77A
A1	Torino	2	C	TORZZZ	BNCPLO77A
A2	Garibaldi	33		GARWWW	RSSMRR75D

Figura 25: Relazione Appartamento con codice fiscale

PROPRIETARIO			
Codice Fiscale	Nome	Cognome	Data Nas
RSSMRR75D	Mario	Rossi	1/1/1975
BNCMRA80F	Maria	Bianchi	10/9/1980
BNCPLO77A	Paolo	Bianchi	3/1/1977

Figura 26: Relazione Proprietario

Siccome non esistono i puntatori bisogna duplicare i valori identificanti nelle due relazioni. Anche questa decomposizione ha dati ridondanti.

4. Separazione in due relazioni distinte per Appartamento e Proprietario con l'aggiunta di un informazione identificante per ciascuna relazione, in questo caso sono:

- codice fiscale nella relazione Appartamento
- codice ecografico nella relazione Proprietario

PROPRIETARIO				
Codice ECO	Codice Fiscale	Nome	Cognome	Data Nas
RMAXXX	RSSMRR75D	Mario	Rossi	1/1/1975
RMAXXX	BNCMRA80F	Maria	Bianchi	10/9/1980
MILYYY	BNCMRA80F	Maria	Bianchi	10/9/1980
MILYYY	BNCPLO77A	Paolo	Bianchi	3/1/1977
TORZZZ	BNCPLO77A	Paolo	Bianchi	3/1/1977
GARWWW	RSSMRR75D	Mario	Rossi	1/1/1975

Figura 27: Relazione Appartamento

APPARTAMENTO				
Categoria	Via	Civico	Sub	Codice ECO
A1	Roma	23	A	RMAXXX
A3	Milano	9		MILYYY
A1	Torino	2	C	TORZZZ
A2	Garibaldi	33		GARWWW

Figura 28: Relazione Proprietario con codice ecografico

Anche in questo caso si presentano le stesse problematiche del punto precedente.

##### 5. Separazione in tre relazione distinte:

- Appartamento
- Persona
- Proprietà (relazione tra Appartamento e Persona)

Per rappresentare la relazione tra Appartamento e Persona si introduce una nuova relazione chiamata Proprietà che contiene i valori degli attributi identificanti di entrambe le relazioni.

APPARTAMENTO				
Categoria	Via	Civico	Sub	Codice ECO
A1	Roma	23	A	RMAXXX
A3	Milano	9		MILYYY
A1	Torino	2	C	TORZZZ
A2	Garibaldi	33		GARWWW

Figura 29: Relazione Appartamento

PROPRIETARIO			
Codice Fiscale	Nome	Cognome	Data Nas
RSSMRR75D	Mario	Rossi	1/1/1975
BNCMRA80F	Maria	Bianchi	10/9/1980
BNCPLO77A	Paolo	Bianchi	3/1/1977

Figura 30: Relazione Persona

Codice ECO	Codice Fiscale
RMAXXX	RSSMRR75D
RMAXXX	BNCMRA80F
MILYYY	BNCMRA80F
MILYYY	BNCPLO77A
TORZZZ	BNCPLO77A
GARWWW	RSSMRR75D

Figura 31: Relazione Proprietà

In questa decomposizione non esistono ridondanze e non si perdono informazioni. Quindi questa è la decomposizione corretta e **senza perdita**. L'unica ridondanza è quella degli attributi identificanti che però è necessaria per rappresentare la relazione tra le due entità.

#### 5.4.5 Proprietà del modello relazionale

- I legami tra relazioni si realizzano attraverso la replicazione di un insieme di attributi, dove il legame tra due tuple si intende stabilito quando esse presentano gli stessi valori negli attributi replicati.
- Il modello relazionale è **value-based** (basato sui valori). Il modello non prevede l'uso di puntatori per rappresentare legami tra i dati. Questo implica:

- Completa indipendenza dall'implementazione e dallo schema fisico
- Facilità di trasferimento dei dati tra sistemi
- Viene rappresentato solo ciò che è rilevante per il sistema informativo

#### 5.4.6 Terminologia

- **Schema di una relazione:** è costituito dal nome della relazione e da un insieme di nomi per i suoi attributi:

$$R(X) \text{ oppure } R(A_1, \dots, A_n) \text{ oppure } R(A_1 : D_1, \dots, A_n : D_n)$$

- **Schema di una base di dati relazionale:** è costituito da un insieme di schemi di relazioni:

$$S = \{R_1(X_1), \dots, R_n(X_n)\}$$

con  $R_1 \neq \dots \neq R_n$

- **Istanza di una relazione di schema  $R(X)$ :** è un insieme  $r$  di tuple su  $X$ :

$$r = \{t_1, \dots, t_k\}$$

- **Istanza di una base di dati relazionale di schema**

$S = \{R_1(X_1), \dots, R_n(X_n)\}$ : è costituito da un insieme di istanze delle relazioni  $R_1(X_1), \dots, R_n(X_n)$ :

$$db = \{r_1, \dots, r_n\}$$

## 5.5 Valori nulli

Dalla definizione di relazione si ha che ogni tupla deve sempre contenere nei propri attributi valori significativi appartenenti ai domini di base del modello. Non sempre però in una base di dati reale esistono i valori per tutti gli attributi di una tupla. Un valore può mancare nei seguenti casi:

- Il valore di un attributo  $A$  è **inesistente** (attributo opzionale a livello concreto), cioè non esiste per questa tupla un valore per l'attributo  $A$ .
- Il valore di un attributo  $A$  è **sconosciuto**, cioè esiste un valore per l'attributo  $A$  di questa tupla ma non è noto alla base di dati.
- Il valore di un attributo  $A$  è inesistente o sconosciuto.

Per poter gestire tali situazioni viene introdotto nel modello relazionale un valore speciale detto **valore nullo**. Quindi gli attributi di una tupla possono assumere un valore del dominio oppure il valore nullo. Il valore nullo non dovrebbe essere abusato.

**Definizione 5.3** (Tupla con valore nullo). Una tupla su  $X$  con valori nullo è definita come una funzione:

$$t : X \rightarrow \{\text{NULL}\} \cup \left( \bigcup_{A \in X} \text{DOM}(A) \right)$$

dove:

$$t[A] = v \in \text{DOM}(A) \vee t[A] = \text{NULL}$$

### 5.5.1 Osservazioni

- La presenza di valori nulli è accettabile **solo in alcuni attributi** (non è possibile avere tuple di soli valori nulli).
- In particolare, negli **attributi replicati per rappresentare legami tra tuple** la presenza di valori nulli può rendere inutilizzabile l'informazione

**Esempio 5.4.** Rappresentare attraverso un insieme di relazioni le informazioni contenute in un orario ferroviario dove si riportino per ogni treno in partenza dalla stazione di Verona Porta Nuova: il numero, l'orario di partenza, la destinazione finale, la categoria, le fermate intermedie (con orario di fermata) e l'orario di arrivo alla destinazione finale.

Lo schema relazionale risultante è il seguente:

TRENO(Numer, Categoria, OraPart, Destinazione, OraArr)  
FERMATA(NumTreno, Stazione, Orario)

In progettazione concettuale equivale a due entità (Treno e Stazione) collegate da una relazione (Fermata).

## 5.6 Vincoli di integrità

Consentono di precisare quali sono le condizioni (vincoli) che le istanze delle relazioni devono rispettare per essere considerate valide.

**Definizione 5.4.** Un vincolo di integrità è una condizione, espressa da un **predicato**, che deve essere **sempre** soddisfatta da **ogni istanza** della base di dati.

**Esempio 5.5.** Consideriamo l'esempio 5.4, i predicati che esprimono possibili

vincoli sono:

$$\begin{aligned}
 & \forall t \in \text{TRENO} : t[\text{OraPart}] \in \{0, 1, \dots, 23\} \\
 & \forall t \in \text{TRENO} : t[\text{MinutoPart}] \in \{0, 1, \dots, 59\} \\
 & \forall t \in \text{TRENO} : t[\text{Numero}] > 0 \\
 & \forall t \in \text{TRENO} : t[\text{Numero}] > 5000 \Rightarrow t[\text{Categoria}] = \text{'Regionale'} \\
 & \forall t, t' \in \text{TRENO} : t \neq t' \Rightarrow t[\text{Numero}] \neq t'[\text{Numero}] \\
 & \forall f \in \text{FERMATA} : \exists t \in \text{TRENO} : f[\text{NumTreno}] = t[\text{Numero}] \\
 & \quad \forall t \in \text{TRENO} : t[\text{Categoria}] = \text{'Regionale'} \\
 & \quad \Rightarrow \exists f \in \text{FERMATA} : f[\text{NumTreno}] = t[\text{Numero}]
 \end{aligned}$$

### 5.6.1 Classificazione

I vincoli di integrità si possono classificare in:

- Vincoli di dominio
- Vincoli di tupla
- Vincoli intrarelazionali
  - Chiavi (vincolo strutturale)
- Vincoli interrelazionali
  - Vincoli di integrità referenziale (vincolo strutturale)

Ci sono anche vincoli che hanno più importanza degli altri in quanto garantiscono **proprietà generali valide per tutti gli schemi relazionali**. Questi vincoli sono detti **vincoli strutturali**.

### 5.6.2 Vincoli di dominio

Impongono una restrizione sui valori che un attributo può assumere. Ad esempio:

$$\begin{aligned}
 & \forall t \in \text{TRENO} : t[\text{OraPart}] \in \{0, 1, \dots, 23\} \\
 & \forall t \in \text{TRENO} : t[\text{MinutoPart}] \in \{0, 1, \dots, 59\} \\
 & \forall t \in \text{TRENO} : t[\text{Numero}] > 0
 \end{aligned}$$

### 5.6.3 Vincoli di tupla

Impongono una restrizione alla **combinazione di valori che una tupla della relazione** può assumere indipendentemente dalle altre tuple, ad esempio:

$$\forall t \in \text{TRENO} : t[\text{Numero}] > 5000 \Rightarrow t[\text{Categoria}] = \text{'Regionale'}$$

#### 5.6.4 Vincoli di intrarelazionali

Impongono una restrizione al contenuto di una relazione e specificano una condizione che **ogni tupla della relazione deve soddisfare rispetto alle altre tuple della medesima relazione**. Una sottocategoria importante di questi vincoli include i **vincoli di chiave**:

- **Superchiave:** Data una relazione di schema  $R(X)$ , un insieme di attributi  $K$  sottoinsieme di  $X$  è **superchiave** per  $R(X)$  se per ogni istanza  $r$  di  $R(X)$  vale la seguente condizione:

$$\forall t, t' \in r : t \neq t' \Rightarrow t[K] \neq t'[K]$$

dove:

$$t[K] \neq t'[K] \equiv \exists A_i \in K : t[A_i] \neq t'[A_i]$$

- **Chiave candidata:** Data una relazione di schema  $R(X)$ , un insieme di attributi  $K$  sottoinsieme di  $X$  è **chiave candidata** (o chiave) per  $R(X)$  se  $K$  è superchiave per  $R(X)$  e vale la seguente condizione:

$$\neg \exists K' \subseteq K : K' \text{ è superchiave per } R(X)$$

**Teorema 5.1.** Esiste sempre una chiave candidata  $K$  per una relazione  $R(X)$

- **Chiave primaria:** Data una relazione di schema  $R(X)$  la sua **chiave primaria** è la chiave candidata scelta per **identificare le tuple della relazione**. La chiave primaria viene usata per rappresentare i legami tra le relazioni e ha le seguenti caratteristiche:

- Non contiene **mai valori nulli**
- Su  $K$  il sistema genera una struttura d'accesso ai dati (o indice) per supportare le interrogazioni

**Esempio 5.6.** Consideriamo la tabella Treno dell'esempio 5.4. I seguenti sottoinsiemi di  $X$  sono:

$K_1 = \{\text{Numero}\}$  è una superchiave e chiave candidata  
 $K_2 = \{\text{Numero}, \text{Categoria}\}$  è una superchiave ma non chiave candidata  
 $K_3 = \{\text{OraPart}, \text{MinutoPart}, \text{Destinazione}, \text{Categoria}\}$   
è una superchiave e chiave candidata

#### 5.6.5 Vincoli interrelazionali

Impongono una restrizione al contenuto di una relazione e specificano una condizione che ogni tupla della relazione deve soddisfare rispetto alle tuple **di altre relazioni della base** di dati.

Una sottocategoria importante di tali vincoli include i **vincoli di integrità referenziale (o vincoli sulle chiavi esportate)**

**Definizione 5.5** (Vincolo di integrità referenziale). Un vincolo di integrità referenziale tra un insieme di attributi  $Y = \{A_1, \dots, A_p\}$  di  $R_1$  e un insieme di attributi  $K = \{K_1, \dots, K_p\}$ , chiave primaria di un'altra relazione  $R_2$ , è soddisfatto se, per ogni istanza  $r_1$  di  $R_1$  e per ogni istanza  $r_2$  di  $R_2$  vale la seguente condizione:

$$\forall t \in r_1 : \exists s \in r_2 : \forall i \in \{1, \dots, p\} : t[A_i] = s[K_i]$$

In presenza di valori nulli (legame opzionale) la condizione diventa:

$$\begin{aligned} \forall t \in r_1 : \exists s \in r_2 : \\ (\forall i \in \{1, \dots, p\} : t[A_i] = s[K_i]) \vee (\exists i \in \{1, \dots, p\} : t[A_i] = \text{NULL}) \end{aligned}$$

### 5.6.6 Notazione per indicare i vincoli strutturali

La chiave primaria di una relazione  $R(X)$  viene indicata sottolineando gli attributi della chiave.

**Esempio 5.7.** Consideriamo l'esempio 5.4. La chiave primaria della tabella Treno è l'attributo Numero:

TRENO(Numero, Categoria, OraPart, Destinazione, OraArr)

La chiave primaria della tabella FERMATA è:

FERMATA(NumTreno, Stazione, Orario)

Il vincolo di integrità referenziale viene indicato in due modi:

- Riquadrando gli attributi soggetti al vincolo e collegando con una freccia il riquadro alla relazione (tabella) da cui la chiave proviene.

**Esempio 5.8.** Consideriamo l'esempio 5.4. Supponendo che esista un vincolo di integrità referenziale sull'attributo NumTreno della tabella FERMATA rispetto alla tabella TRENO, la notazione per indicarlo è la seguente:

$\rightarrow$ TRENO(Numero, Categoria, OraPart, Destinazione, OraArr)

FERMATA([NumTreno], Stazione, Orario)

- Indicare con una freccia l'attributo soggetto al vincolo verso la relazione da cui la chiave proviene, ad esempio:

$$R_1, D \rightarrow R_2 \\ R_3(Y, X) \rightarrow R_1$$

L'attributo opzionale è indicato con un asterisco accanto al nome dell'attributo.

## 5.7 Progettazione logica dallo schema concettuale

L'obiettivo della progettazione logica è quello di produrre uno schema logico che descriva in modo **corretto ed efficace** tutte le informazioni contenute nello schema concettuale.

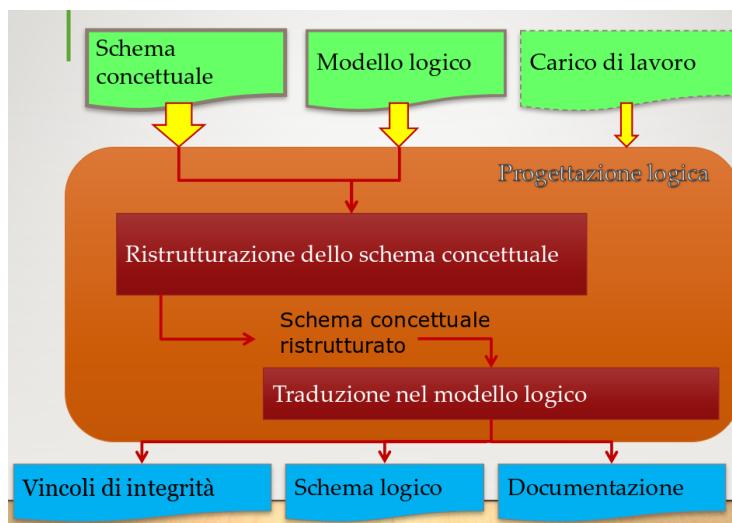


Figura 32: Diagramma della progettazione logica

### 5.7.1 Ristrutturazione dello schema concettuale

La ristrutturazione dello schema concettuale può essere necessaria per i seguenti motivi:

- Semplificare la fase successiva di traduzione
- Ottimizzare le strutture rispetto al carico di lavoro che è composto da:
  - Dimensioni dei dati
  - Caratteristiche delle operazioni
    - \* Costo (numero di accessi alla memoria secondaria)
    - \* Frequenza

Si divide nelle seguenti fasi:

1. **Analisi delle ridondanze dovute a presenza di dati derivabili:**

- La derivabilità è la presenza di dati derivabili da altri:
  - Attributi derivabili da altri attributi della stessa entità (o relazione)
  - Attributi derivabili da attributi di altre entità (o relazioni)
  - Attributi derivabili dal conteggio di occorrenze
- L'obiettivo dell'analisi delle ridondanze è quello di analizzare i dati derivabili presenti nello schema (o richiesti dalle operazioni) e decidere se:
  - Mantenere il dato derivabile (memorizzandolo esplicitamente). Questo conviene quando il costo del calcolo è maggiore del costo di memorizzazione.
  - Calcolare il dato derivabile quando serve. Questo conviene quando il costo di memorizzazione è maggiore del costo del calcolo.

Il costo viene misurato come il numero di accessi per unità di tempo in quanto vengono considerate le frequenze di calcolo e di aggiornamento del dato.

## 2. Eliminazione delle generalizzazioni:

L'obiettivo è quello di sostituire le **generalizzazioni** presenti nello schema ER con strutture alternative, in quanto non direttamente rappresentabili nel modello relazionale. Esistono tre possibili soluzioni:

- (a) Accorpamento delle entità figlie nel padre: I passi sono i seguenti.
  - i. Eliminazione delle entità figlie
  - ii. Accorpamento nel padre di tutti gli attributi specifici delle entità figlie come attributi opzionali
  - iii. Accorpamento nel padre delle relazioni che coinvolgono le entità figlie assegnando cardinalità minima uguale a zero (lato entità padre)
  - iv. Aggiunta di un attributo "tipo" per distinguere nel padre le occorrenze delle entità figlie eliminate

**Esempio 5.9.** Consideriamo il seguente esempio:

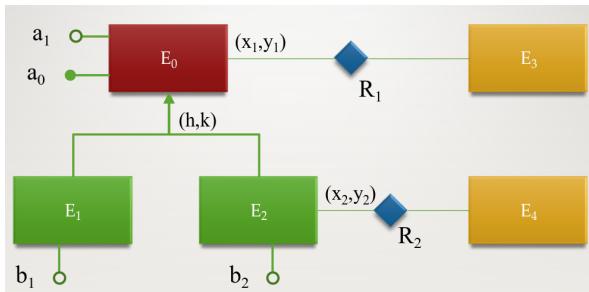


Figura 33: Esempio di accorpamento delle entità figlie nel padre

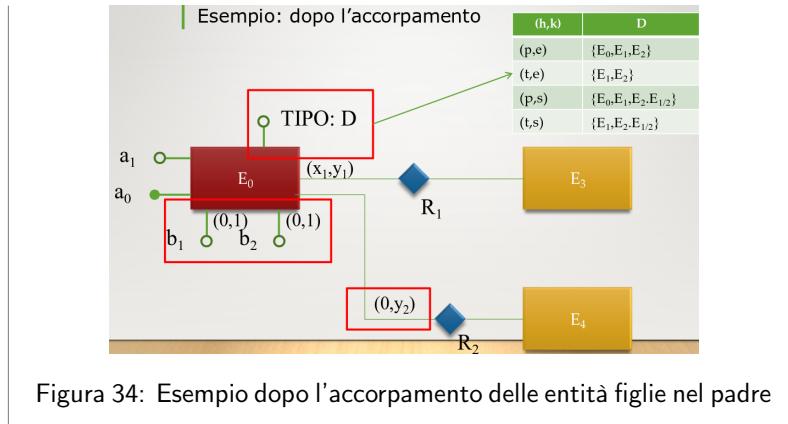


Figura 34: Esempio dopo l'accorpamento delle entità figlie nel padre

- (b) Accorpamento del padre nelle entità figlie: La precondizione per poter applicare questa trasformazione è che la generalizzazione sia **totale**. I passi sono i seguenti.
- Eliminazione dell'entità padre
  - Replicazione degli attributi dell'entità padre su ogni entità figlia
  - Partizionamento sui figli delle relazioni che coinvolgono l'entità padre assegnando cardinalità minima uguale a zero (lato entità esterne)

**Esempio 5.10.** Consideriamo il seguente esempio:

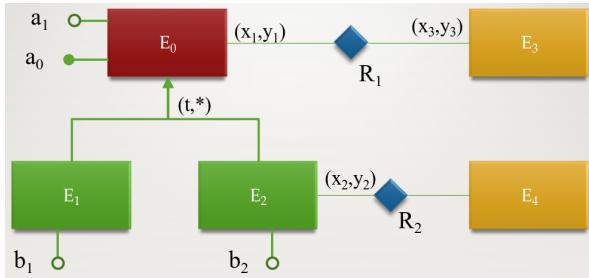


Figura 35: Esempio di accorpamento del padre nelle entità figlie

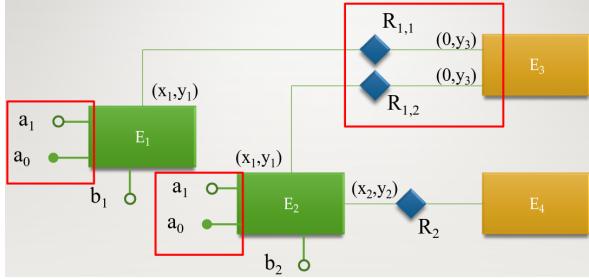


Figura 36: Esempio dopo l'accorpamento del padre nelle entità figlie

- (c) Sostituzione della generalizzazione con relazioni: I passi sono i seguenti.
- Eliminazione della generalizzazione
  - Inserire  $n$  relazioni tra il padre e ciascuna delle  $n$  entità figlie
  - Tutti gli attributi conservano la loro collocazione

**Esempio 5.11.** Consideriamo il seguente esempio:

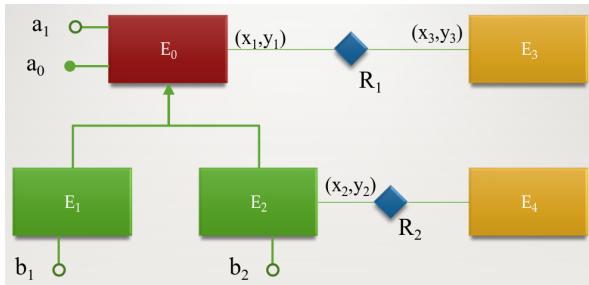


Figura 37: Esempio di accorpamento di entità e relazioni

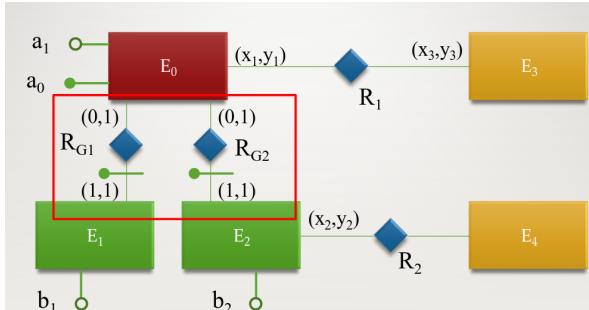


Figura 38: Esempio dopo l'accorpamento di entità e relazioni

### 3. Accorpamento (o partizionamento) di entità e relazioni:

- Partizionamento di entità:  
È la sostituzione di un'entità dello schema con una coppia di entità con

lo stesso identificatore e una relazione uno a uno che le lega tra loro. Si esegue solo nel caso in cui esistano operazioni frequenti che trattano solo un sottoinsieme degli attributi dell'entità da partizionare

**Esempio 5.12.** Consideriamo il seguente esempio:

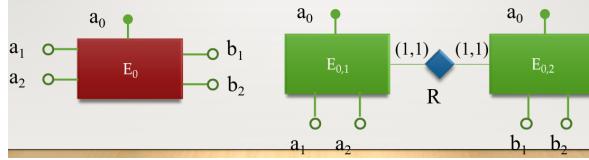


Figura 39: Esempio di accorpamento di entità

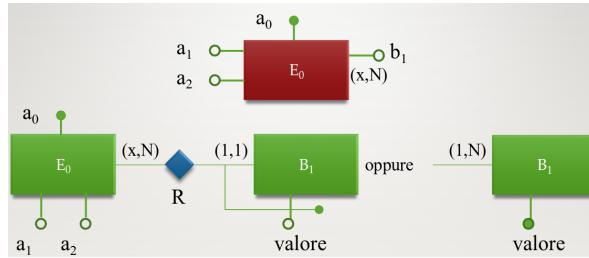


Figura 40: Eliminazione di attributi multivalore

- Partizionamento di relazioni:

Si realizza sostituendo una relazione  $R$  con due relazioni distinte  $R_1$  e  $R_2$ , dove le occorrenze di  $R$  si partizionano tra  $R_1$  e  $R_2$ . È conveniente quando esistono operazioni che si riferiscono ad un sottoinsieme delle occorrenze di  $R$  e altre operazioni che si riferiscono alle altre occorrenze di  $R$ .

- Scelta degli identificatori principali: Va scelto un identificatore principale per ogni entità in quanto tale identificatore verrà utilizzato nel modello relazionale per identificare le tuple che rappresentano istanze di entità. Si scelgono seguendo i seguenti criteri:

- Gli identificatori che includono attributi optional non possono essere chiavi primarie
- Meglio identificatori con pochi attributi
- Meglio identificatori utilizzati nelle operazioni

### 5.7.2 Traduzione nel modello logico

La traduzione è un processo automatico che si realizza applicando allo schema concettuale ristrutturato un insieme di regole di traduzione. **Ogni regola si applica ad un costrutto del modello concettuale ER e produce una o più strutture del modello relazionale.**

- Si rappresenta un'istanza di entità con una tupla.

- Si rappresenta un'istanza di relazione con un legame tra tuple o con una tupla esplicita.
- Si rappresenta l'identificatore principale di un'entità con la chiave primaria della tabella che implementa l'entità.

Le relazioni binarie del modello ER sono classificate in:

- **Uno a uno:** se entrambe le entità coinvolte nella relazione hanno cardinalità massima uguale a 1.
- **Uno a molti:** se una delle entità coinvolte nella relazione ha cardinalità massima uguale a 1 e l'altra ha cardinalità massima maggiore di 1.
- **Molti a molti:** se entrambe le entità coinvolte nella relazione hanno cardinalità massima maggiore di 1.

Le regole di traduzione sono le seguenti:

### 1. Entità:

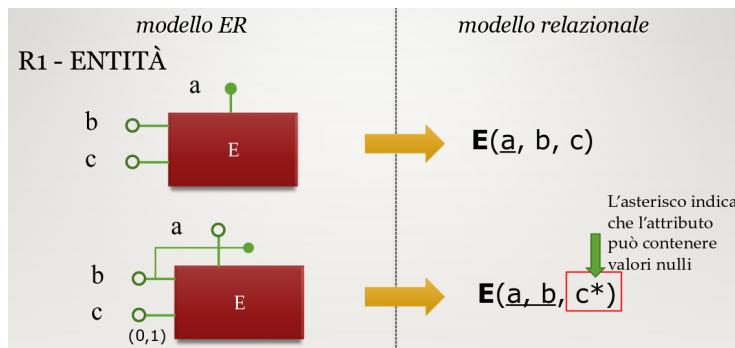


Figura 41: Regola di traduzione per entità

### 2. Relazione uno a molti:

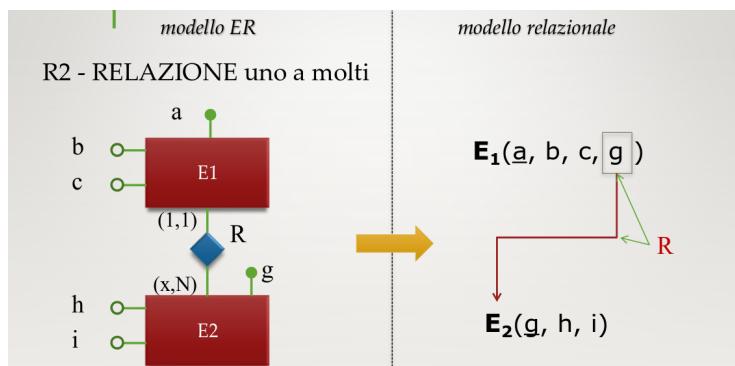


Figura 42: Regola di traduzione per relazione uno a molti

### 3. Relazione uno a uno:

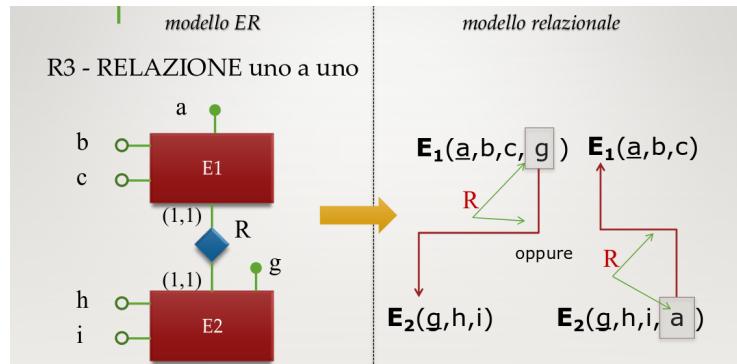


Figura 43: Regola di traduzione per relazione uno a uno

### 4. Relazione molti a molti:

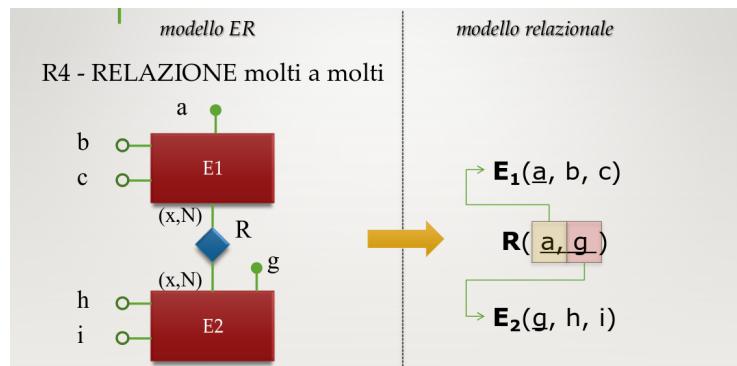


Figura 44: Regola di traduzione per relazione molti a molti

### 5. Relazione uno a molti con identificatore esterno:

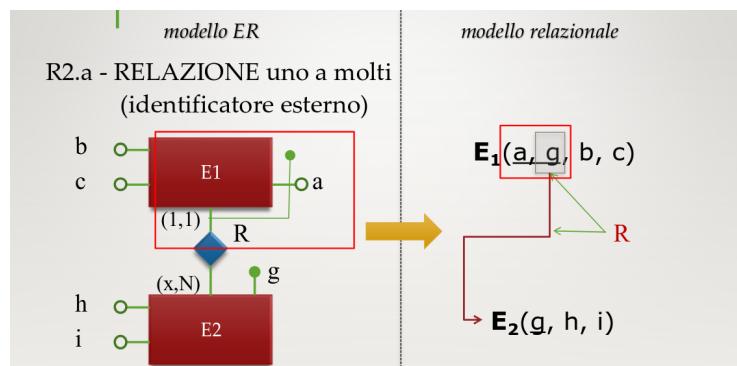


Figura 45: Regola di traduzione per relazione uno a molti con identificatore esterno

### 6. Relazione uno a molti con attributo sulla relazione:

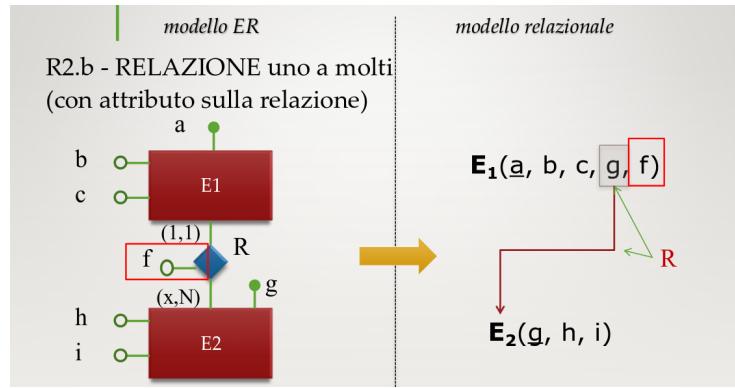


Figura 46: Regola di traduzione per relazione uno a molti con attributo sulla relazione

#### 7. Relazione uno a uno con cardinalità minima uguale a 0:

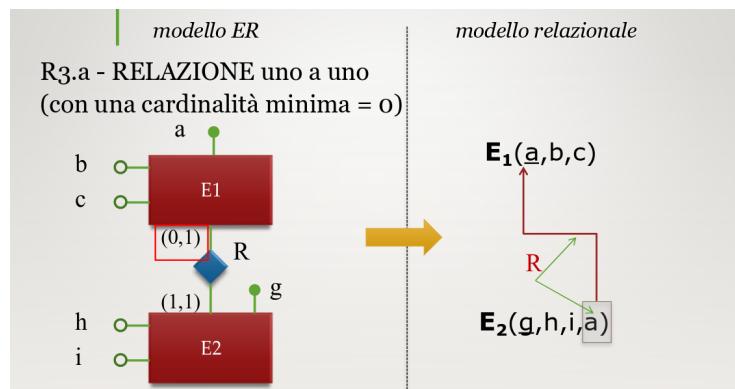


Figura 47: Regola di traduzione per relazione uno a uno con cardinalità minima uguale a 0

#### 8. Relazione uno a uno con due cardinalità minime uguali a 0:

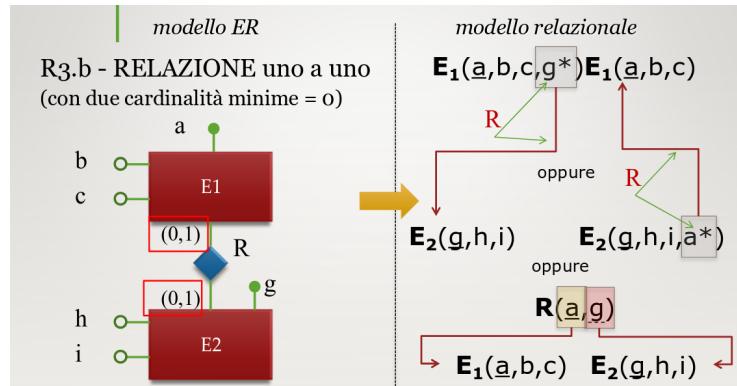


Figura 48: Regola di traduzione per relazione uno a uno con due cardinalità minime uguali a 0

#### 9. Relazione molti a molti con attributi:

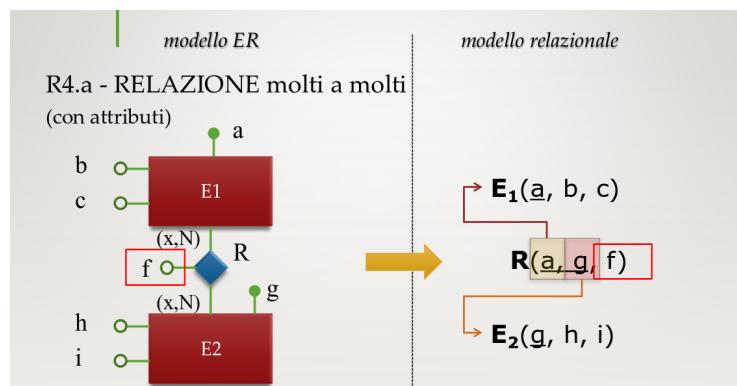


Figura 49: Regola di traduzione per relazione molti a molti con attributi

#### 10. Relazione molti a molti con identificatori da più attributi:

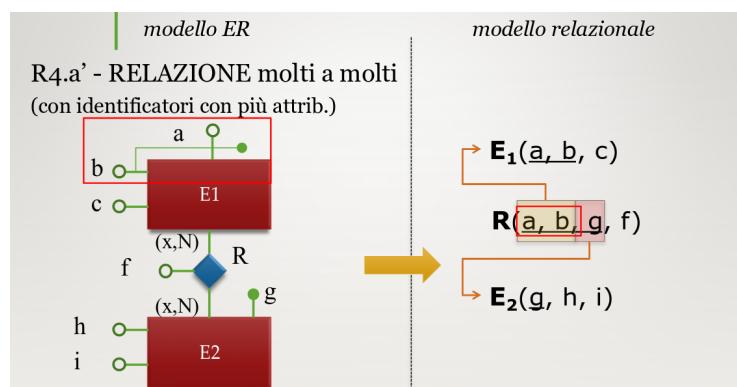


Figura 50: Regola di traduzione per relazione molti a molti con identificatori da più attributi

### 11. Relazione molti a molti con relazione ternaria:

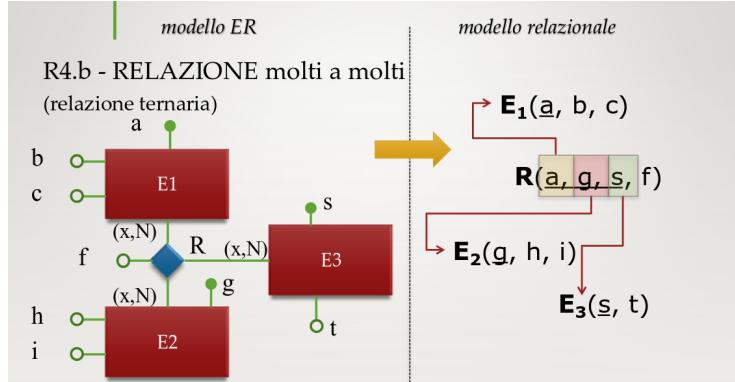


Figura 51: Regola di traduzione per relazione molti a molti con relazione ternaria

### 12. Relazione molti a molti con relazione ternaria a cardinalità (1,1):

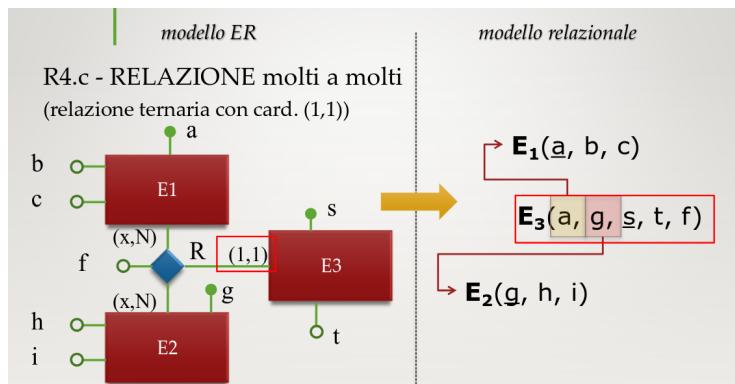


Figura 52: Regola di traduzione per relazione molti a molti con relazione ternaria a cardinalità (1,1)

## 6 Tecnologie NoSQL: sistemi document based

Ci sono diversi approcci NoSQL per la gestione dei dati:

- **Sistemi key-value:** tutti i dati sono rappresentati per mezzo di **coppie chiave-valore**, dove la chiave identifica le istanze e il valore può avere una qualsiasi struttura anche non omogenea nella collezione. Un esempio di questo approccio è il sistema HADOOP.
- **Sistemi document-store:** i dati sono rappresentati come **collezioni di documenti** (oggetti con struttura complessa). Ogni oggetto della collezione ha una struttura complessa (con dati **incapsulati**) senza schema fisso. Consentono la definizione di indici secondari su proprietà degli oggetti (attributi comuni). Un esempio di questo approccio è il sistema MongoDB.

- **Sistemi extensible-record-store o column store:** i dati sono rappresentati da **collezioni di record**, dette tabelle, dove ogni record ha un certa lista di colonne che può variare da record a record (struttura variabile). Esempi di sistemi di questa categoria sono: BigTable di Google, HBase, Cassandra.
- **Sistemi graph-store:** le collezioni di dati sono rappresentati come **grafi**. I nodi rappresentano gli oggetti, gli archi le relazioni o proprietà (approccio RDF - Resource Description Framework). Esempi di sistemi di questa categoria sono: Neo4j e OrientDB

## 6.1 Modello dei dati dei sistemi document-store

Le caratteristiche principali di tali modelli sono:

- I dati sono rappresentati da **collezioni di oggetti** (detti documenti)
- Gli oggetti hanno **struttura complessa** (non sono tuple piatte ma contengono dati nidificati)
- L'incapsulamento (encapsulation) dei dati è un aspetto chiave di questi modelli
- Gli oggetti di una collezione **non devono avere necessariamente tutti la stessa struttura**, tuttavia dovrebbero condividere un **nucleo di proprietà comuni**

### 6.1.1 Progettazione dei dati come documenti

Le decisioni chiave nel processo di progettazione dei dati usando questo approccio sono:

- Scegliere la **chiave di accesso**
- **Strutturare i documenti** (grado di incapsulamento)
- **Rappresentazione delle relazioni** (legami) tra i documenti

Esistono in particolare due approcci per rappresentare le relazioni tra i documenti:

- Attraverso **riferimenti** che si ottengono esportando l'ID del documento
- Attraverso **documenti nidificati** (incapsulamento)

**Esempio 6.1.** L'uso di riferimenti per rappresentare i legami tra documenti porta a **schemi normalizzati**.

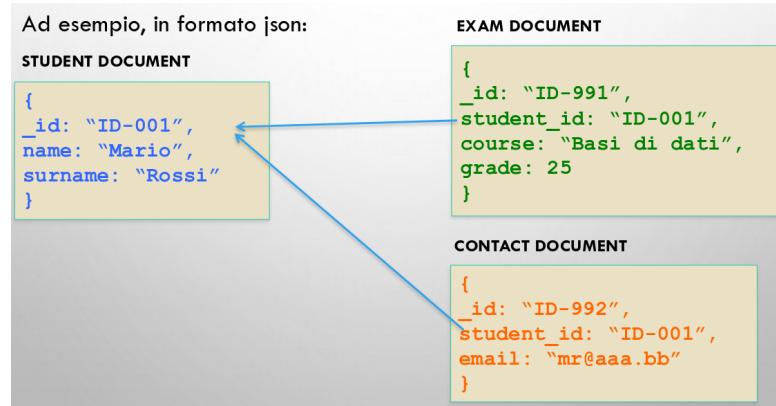


Figura 53: Esempio di documenti con riferimenti

Un'alternativa è quella in cui si rappresentano i legami tra documenti attraverso documenti nidificati:

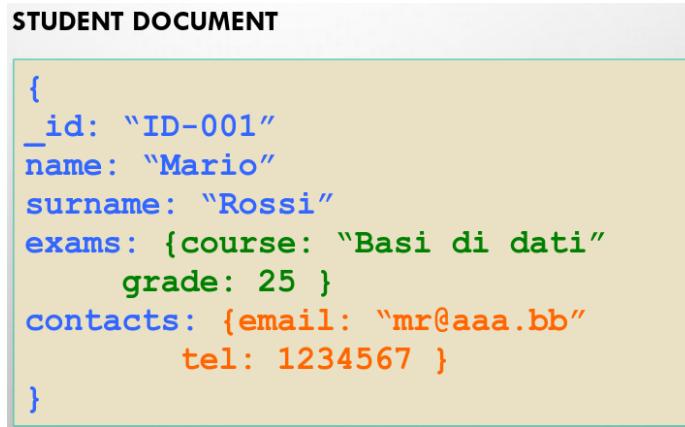


Figura 54: Esempio di documenti con documenti nidificati

Questa soluzione è meno efficiente.

- L'approccio normalizzato consente di rappresentare il dato in modo simile a quanto avviene nel modello relazionale. Non c'è ridondanza e gli aggiornamenti sono più semplici (riguardano sempre una sola istanza).

**Esempio 6.2.** Un esempio dell'approccio normalizzato è il seguente:

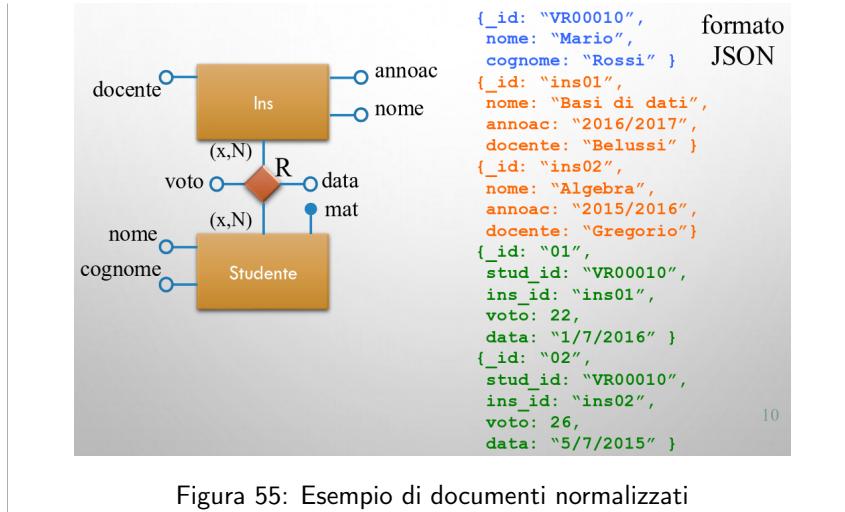


Figura 55: Esempio di documenti normalizzati

- L'approccio embedded consente di rappresentare in un **unico documento** quanto viene rappresentato di solito nel modello ER con una entità E insieme alle sue "entità deboli" (entità collegate ad E attraverso identificatori esterni). Vale a dire è possibile rappresentare nello stesso documento un'istanza di informazione con altre istanze di informazione in essa logicamente contenute. In questo caso il documento può essere aggiornato come oggetto complesso in modo atomico.

### 6.1.2 Rappresentazione di un documento

Per rappresentare un documento si segue lo standard JSON (JavaScript Object Notation):

```
{
  _id: "ID-001",
  name: "Mario",
  surname: "Rossi",
  exams: [{course: "Basi di dati",
            grade: 25 },
           {course: "Programmazione I",
            grade: 27 }],
  contacts: {email: "mr@aaa.it",
             tel: ["1234567", "345237"] }
}
```

Figura 56: Struttura di un documento JSON

- Le parentesi graffe delimitano il documento
- Il documento contiene una lista di proprietà separate da virgolette
- Ogni proprietà si specifica con la sintassi:

nome\_proprietà: valore

- I valori atomici si indicano come costanti stringa (delimitate da apici) o numeri interi o reali
- Le proprietà multi valore si specificano elencando i valori tra parentesi quadre
- Il valore di una proprietà può essere un documento annidato che inizia e finisce con parentesi graffe
- Le proprietà multi valore possono contenere anche liste di documenti

## 6.2 Traduzione di uno schema ER in collezioni di documenti

Per procedere con la progettazione logica dei dati come documenti bisogna fare delle ipotesi di partenza:

- Supponiamo di applicare un **approccio embedded**
- Supponiamo che lo schema ER sia già stato ristrutturato e quindi sia **privò di generalizzazioni**
- Vengono eliminate tutte le **relazioni ternarie (o ennarie)**
- A partire dalle operazioni di elaborazione da eseguire sui dati vengono identificate le collezioni di documenti da generare nel sistema:

$$\{\text{DOC}_1, \dots, \text{DOC}_N\}$$

possono essere al massimo una per ogni entità dello schema.

### 6.2.1 Regole di traduzione

- Ogni collezione  $\text{DOC}_i$  è in corrispondenza con un'entità EP dello schema ER, detta **entità principale della collezione di documenti**
- Per ogni istanza dell'entità principale ci sarà uno e sun solo documento nella collezione
- Ogni istanza delle **entità non principali** sarà rappresentata da uno (o più) documenti **incapsulati in un altro documento** che rappresenta un'istanza di un'entità principale
- Le istanze di **relazione dello schema ER** saranno rappresentate **strutturalmente** (attraverso l'incapsulamento) oppure esportando ID di documento o chiavi applicative

### 6.2.2 Etichettatura dello schema ER

Ci sono diverse fasi per tradurre uno schema ER in collezioni di documenti:

#### 1. Individuazione delle entità principali

- (a) Etichettare l'entità principale di una collezione di documenti  $\text{DOC}_i$  con l'etichetta  $\text{DOC}_i$

**Esempio 6.3.** Ad esempio, se la collezione  $DOC_3$  ha come entità principale l'entità  $E_2$ , questa va etichettata come segue:

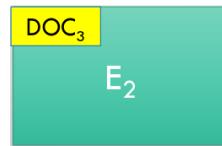


Figura 57: Esempio di etichettatura dell'entità principale

## 2. Incapsulamento delle entità non principali

(a) Per ogni entità  $E_i$  non etichettata come principale, va deciso **in quale collezione incapsularla**. Tutti i casi possibili sono:

- i. Se  $E_i$  è collegata attraverso una relazione  $R$  ad una sola entità principale  $EP$  allora  $E_i$  verrà incapsulata in  $EP$
- ii. Se  $E_i$  è collegata attraverso le relazioni  $R_1, \dots, R_N$  a più entità principali  $EP_1, \dots, EP_N$  bisogna scegliere in quale entità principale  $EP_x$  incapsulare  $E_i$ .  $E_i$  verrà quindi incapsulata in  $EP_x$
- iii. Se  $E_i$  non è collegata attraverso una relazione  $R$  ad un'entità principale, ma è collegata ad altre entità  $E_{Y_1}, \dots, E_{Y_M}$  già collegate ad altre entità principali, bisogna scegliere l'entità  $E_{Y_K}$  in cui incapsulare  $E_i$ .  $E_i$  verrà quindi incapsulata in  $E_{Y_K}$

Per rappresentare che  $E_i$  viene incapsulata in  $EP$  si usa una freccia che indica la direzione dell'incapsulamento:

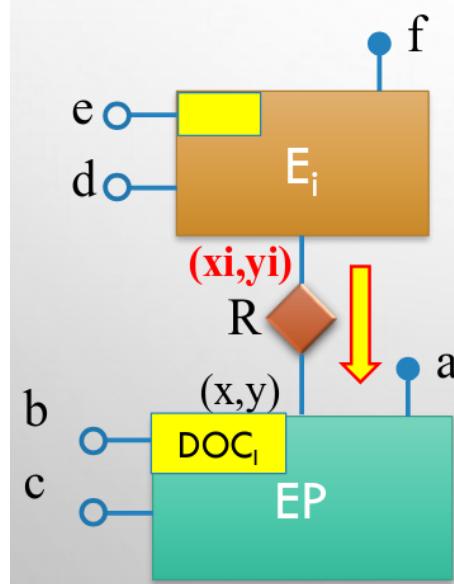


Figura 58: Esempio di freccia per indicare l'incapsulamento

L'etichetta da assegnare all'entità  $E_i$  dipende dalla cardinalità di R lato  $E_i$

- Se  $E_i$  è collegata attraverso una relazione R ad un'entità principale EP (unica scelta tra quelle possibili).  $E_i$  verrà quindi incapsulata in EP.  $E_i$  partecipa alla relazione R con cardinalità massima (1, 1).

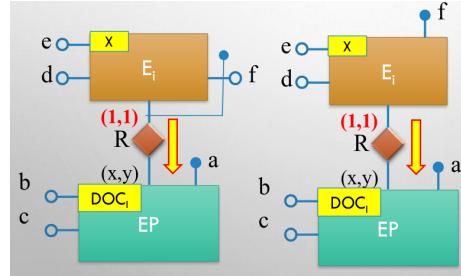


Figura 59: Esempio di incapsulamento con cardinalità (1,1)

Se la relazione R è uno a uno o uno a molti con (1, 1) lato  $E_i$ , allora l'etichetta di  $E_i$  è uguale a "X" e indica che l'incapsulamento è senza perdita e non ridondante

- Se  $E_i$  è collegata attraverso una relazione R ad un'entità principale EP (unica o a scelta).  $E_i$  verrà quindi incapsulata in EP.  $E_i$  partecipa alla relazione R con cardinalità massima (0, 1).

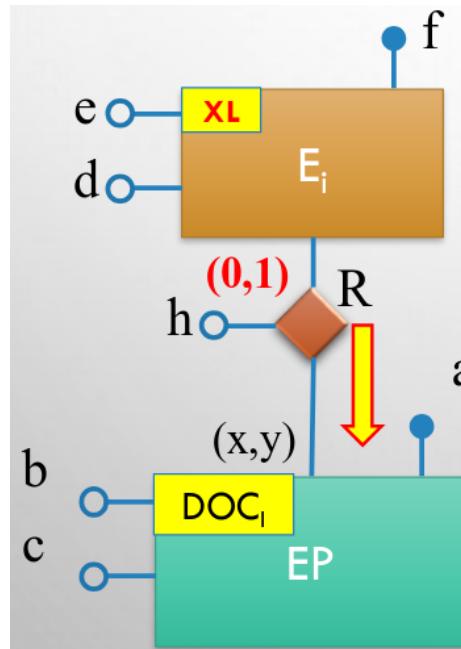


Figura 60: Esempio di incapsulamento con cardinalità (0,1)

Se la relazione R è uno a uno o uno a molti con (0, 1) lato  $E_i$ , allora l'etichetta di  $E_i$  è uguale a "XL" e indica che **si perdonano** le istanze

di  $E_i$  che non sono collegate a nessuna istanza di EP. Ciò viene indicato dal suffisso "L" (loss) aggiunto all'etichetta.

- iii. Se  $E_i$  è collegata attraverso una relazione R ad un'entità principale EP (unica o a scelta).  $E_i$  verrà quindi incapsulata in EP.  $E_i$  partecipa alla relazione R con cardinalità massima (1, N).

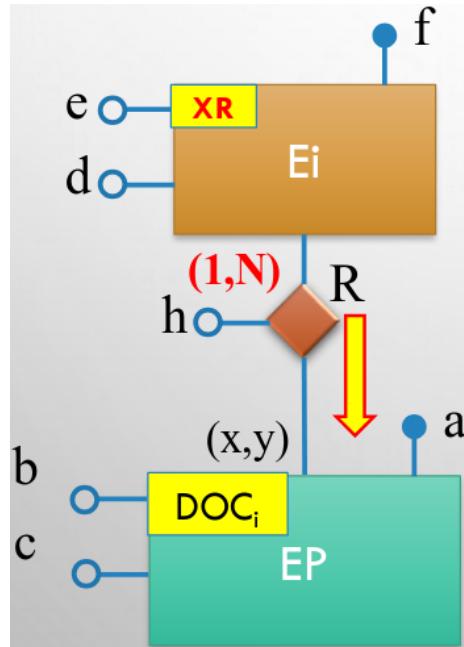


Figura 61: Esempio di incapsulamento con cardinalità (1,N)

Se la relazione R è molti a molti o molti a uno con (1, N) lato  $E_i$ , allora le istanze di  $E_i$  vengono rappresentate in modo **ridondante** in quanto sono collegate a più istanze di EP. L'etichetta di  $E_i$  è uguale a "XR" e indica che l'incapsulamento è ridondante dal suffisso "R".

- iv. Se  $E_i$  è collegata attraverso una relazione R ad un'entità principale EP (unica o a scelta).  $E_i$  verrà quindi incapsulata in EP.  $E_i$  partecipa alla relazione R con cardinalità massima (0, N).

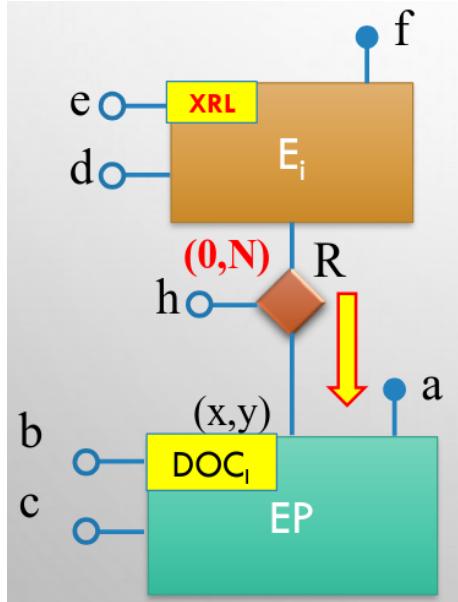


Figura 62: Esempio di incapsulamento con cardinalità  $(0,N)$

Se la relazione  $R$  è molti a molti o molti a uno con  $(0, N)$  lato  $E_i$ , allora le istanze di  $E_i$  vengono rappresentate in modo **ridondante** in quanto sono collegate a più istanze di  $EP$  e si **perdonano** quelle non collegate. Ciò viene indicato dall'etichetta di  $E_i$  uguale a "XLR" che indica che l'incapsulamento è ridondante e con perdita, dal suffisso "LR".

- v. Se  $E_i$  è collegata attraverso una relazione  $R$  ad un'entità  $E_Y$  (unica o a scelta) a sua volta collegata ad un'entità principale  $EP$  allora  $E_i$  verrà incapsulata in  $E_Y$ .

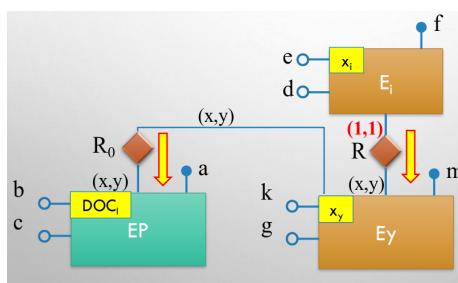


Figura 63: Esempio di incapsulamento in un'entità non principale

Se la relazione  $R$  è uno a uno o uno a molti con  $(1, 1)$  lato  $E_i$ , allora l'etichetta di  $x_i$  è uguale all'etichetta  $x_y$ . Se invece di  $(1, 1)$  ci sono altri vincoli si ha:

- $(0, 1) \rightarrow x_y L$
- $(1, N) \rightarrow x_y R$
- $(0, N) \rightarrow x_y LR$

### 3. Rappresentazione delle relazioni non già rappresentate strutturalmente:

Tutte le relazioni che sono state etichettate da una freccia piena sono già rappresentate strutturalmente. Quelle che rimangono devono essere rappresentate con il meccanismo dei riferimenti

- Caso 1:

Per ogni relazione binaria  $R$  che coinvolge due entità principali  $EP_1$  ed  $EP_2$ , va deciso in quale collezione incapsularla. Tutti i casi possibili sono:

- (a)  $R$  viene incapsulata in una sola entità principale  $EP_1$  o  $EP_2$
- (b)  $R$  viene incapsulata in entrambe le entità principali  $EP_1$  e  $EP_2$   
(questa scelta genera ridondanza)

Viene utilizzata una freccia semplice per indicare dove viene rappresentata la relazione e la sua etichetta indica se la rappresentazione è per id del documento (ID) o per chiave applicativa (KEY). Un suffisso "(R)" indica che la rappresentazione è ridondante

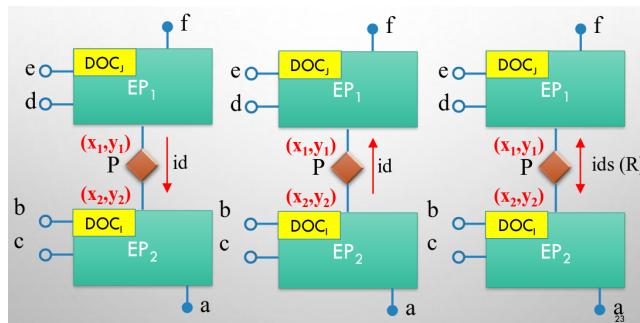


Figura 64: Esempio di etichettatura di relazioni binarie

#### Casi particolari:

- Un'entità non principale  $E_i$  incapsulata in un'entità principale  $EP_1$  è collegata attraverso una relazione binaria  $R$  ad un'altra entità principale  $EP_2$ .

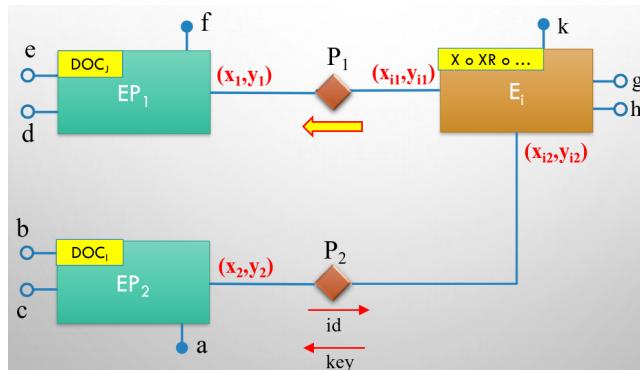


Figura 65: Esempio di etichettatura di relazioni binarie con entità non principale

- Un'entità non principale  $E_i$  (incapsulata in un'entità principale  $EP_1$ ) è collegata attraverso una relazione binaria  $R$  ad un'altra entità non principale  $E_Y$  (incapsulata in un'altra entità principale  $EP_2$ ).

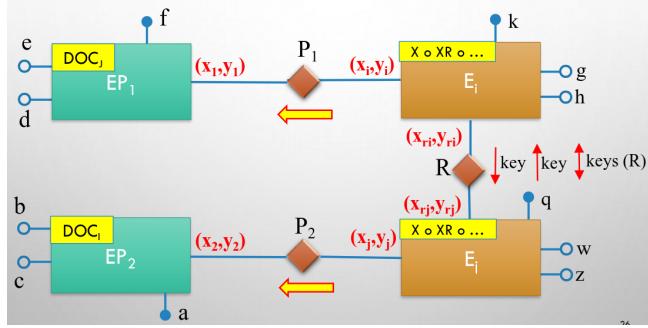


Figura 66: Esempio di etichettatura di relazioni binarie con entità non principale

#### 4. Regole di traduzione di uno schema etichettato:

Le strutture dei documenti vengono descritte con una sintassi semplificata e la generazione della struttura dipende poi dallo specifico sistema scelto

- (a) Regola 1: Entità principale



Figura 67: Regola di traduzione per entità principale

<TIPO> può essere solo:

- string
- integer
- number
- boolean
- date
- time
- timestamp

L'attributo `_id` è sempre presente ed è autogenerato dal sistema. In tutti gli esempi a seguire i tipi sono stati scelti a caso. Nel caso specifico si scelgono i tipi in base al contesto applicativo (requisiti)

- (b) Regola 2: Entità non principale

- i. Se c'è un'entità principale EP con entità debole  $E_i$  (o non principale) da incapsulare, allora  $E_i$  partecipa alla relazione R con vincolo di cardinalità (1, 1) e EP con vincolo di cardinalità (X, N):

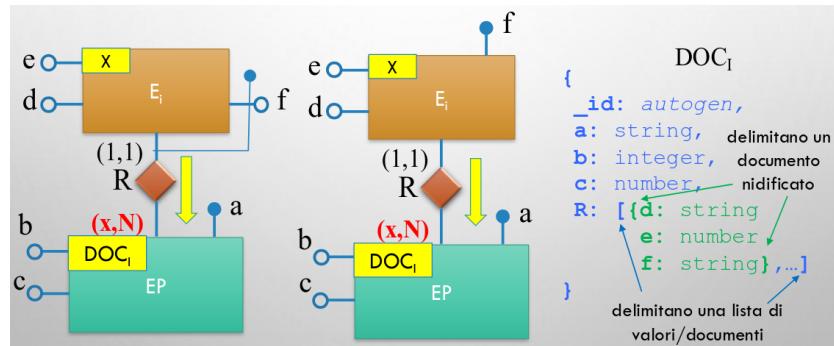


Figura 68: Regola di traduzione per entità principale con entità debole

**Esempio 6.4.** Un esempio di applicazione è il seguente:

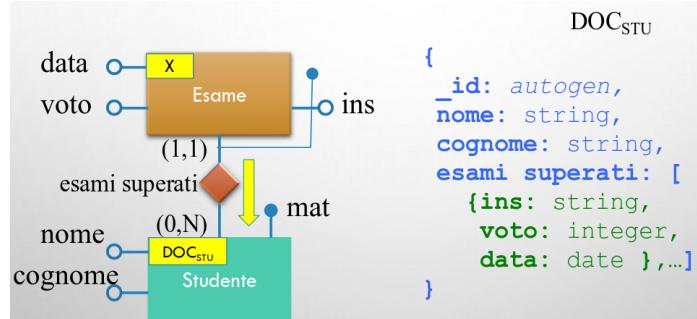


Figura 69: Esempio di traduzione di entità principale con entità debole

- ii. Se c'è un entità principale EP con entità debole  $E_i$  (o non principale) da incapsulare, allora  $E_i$  partecipa alla relazione R con vincolo di cardinalità (1, 1) e EP con vincolo di cardinalità (X, 1):

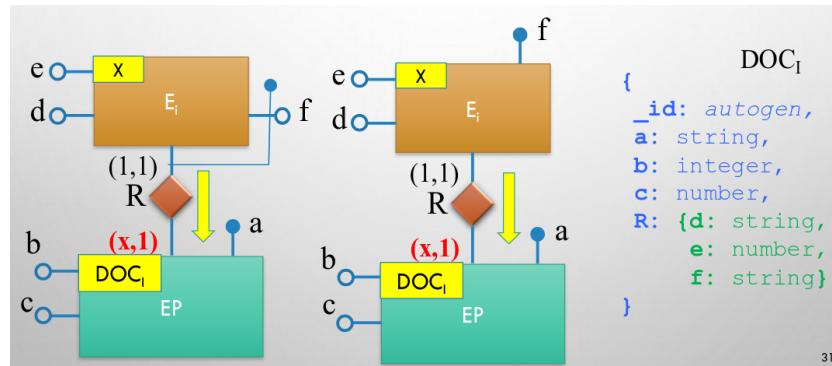


Figura 70: Regola di traduzione per entità principale con entità debole

(c) Regola 3: Entità debole o non principale

- i. Se c'è un'entità principale EP con entità debole E<sub>i</sub> (o non principale) da incapsulare, allora E<sub>i</sub> partecipa alla relazione R con vincolo di cardinalità (0, 1) o (0, N) o (1, N) e EP con vincolo di cardinalità (X, N):

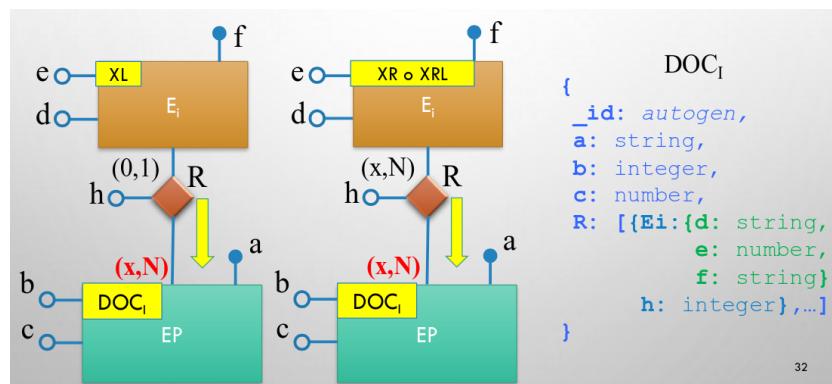


Figura 71: Regola di traduzione per entità principale con entità debole

- ii. Se c'è un'entità principale EP con entità debole E<sub>i</sub> (o non principale) da incapsulare, allora E<sub>i</sub> partecipa alla relazione R con vincolo di cardinalità (0, 1) o (0, N) o (1, N) e EP con vincolo di cardinalità (X, 1):

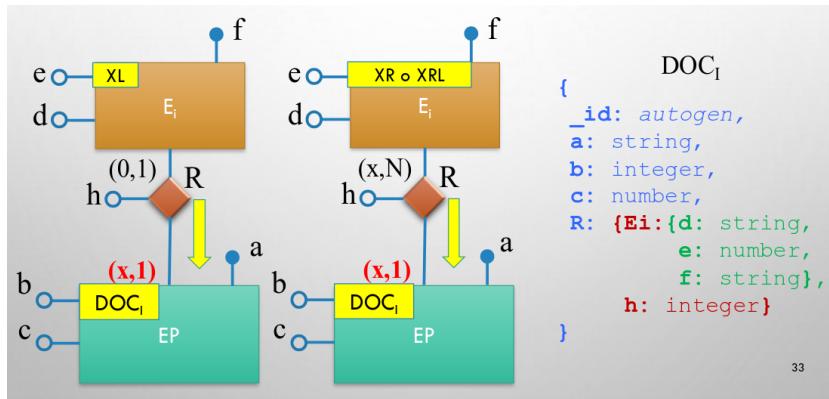


Figura 72: Regola di traduzione per entità principale con entità debole

(d) Regola 4: Entità non principale incapsulata in un'entità non principale

- i. Se  $E_i$  è collegata attraverso una relazione  $R$  ad un'entità  $E_y$  (unica o a scelta) a sua volta collegata con un'entità EP, allora  $E_i$  viene incapsulata in  $E_y$  e  $E_y$  viene incapsulata in EP. I vincoli su  $E_y$  e EP sono  $(0, N)$  o  $(1, N)$ :

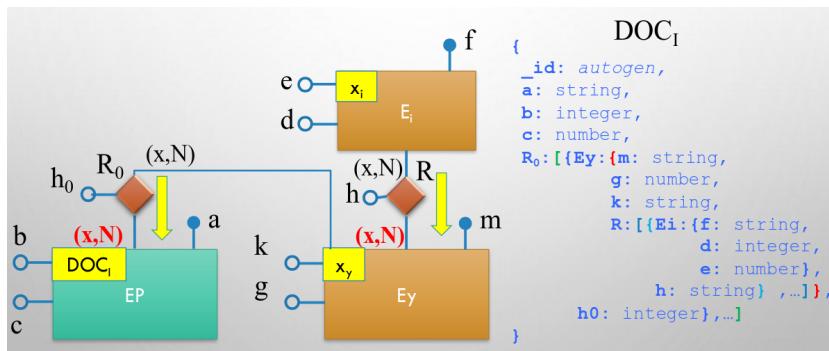


Figura 73: Regola di traduzione per entità incapsulata in un'entità non principale

- ii. Se  $E_i$  è collegata attraverso una relazione  $R$  ad un'entità  $E_y$  (unica o a scelta) a sua volta collegata con un'entità EP, allora  $E_i$  viene incapsulata in  $E_y$  e  $E_y$  viene incapsulata in EP. I vincoli su  $E_y$  e EP sono  $(0, 1)$  o  $(1, 1)$ :

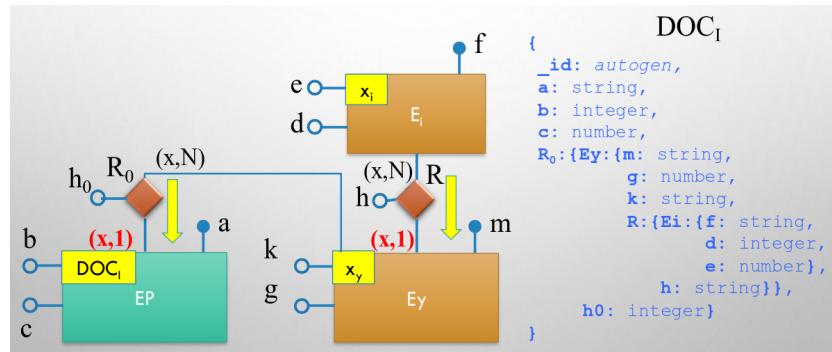


Figura 74: Regola di traduzione per entità incapsulata in un'entità non principale

(e) Regola 5: Relazione molti a molti tra entità principali

Se c'è una relazione molti a molti tra entità principali EP<sub>1</sub> ed EP<sub>2</sub>, allora EP<sub>2</sub> partecipa alla relazione R con vincolo di cardinalità (0, N) o (1, N) e incapsula R solo in EP<sub>2</sub>:

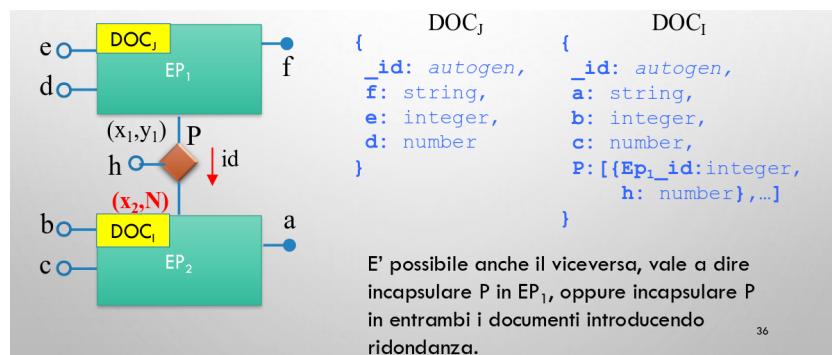


Figura 75: Regola di traduzione per relazione molti a molti tra entità principali

**Esempio 6.5.** Un esempio di applicazione è il seguente:

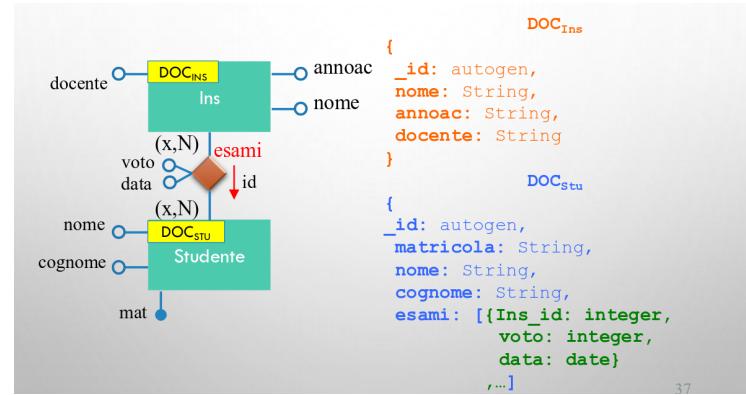


Figura 76: Esempio di traduzione di relazione molti a molti tra entità principali

oppure

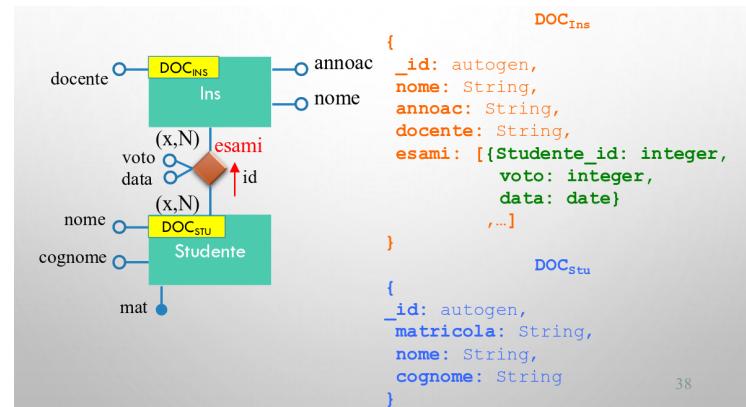


Figura 77: Esempio di traduzione di relazione molti a molti tra entità principali

(f) Regola 6: Relazione binaria tra entità non principale ed entità principale

- i. Relazione binaria  $P_2$  tra un'entità non principale  $E_i$ , da incapsulare in  $EP_1$ , e un'entità principale  $EP_2$  con vincolo  $(X, 1)$  lato  $E_i$ :

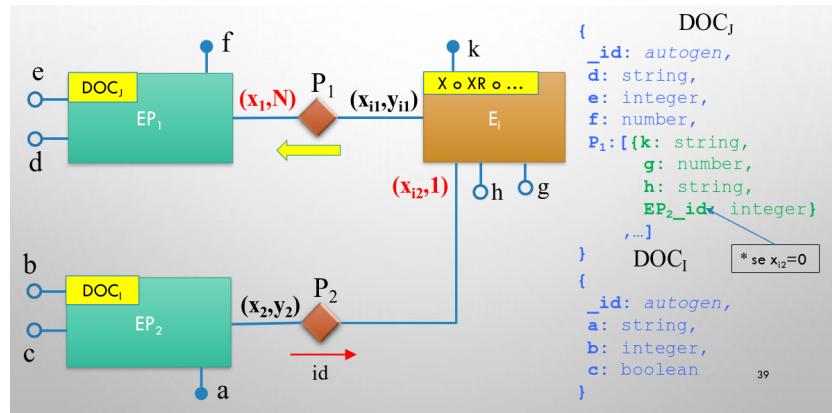


Figura 78: Regola di traduzione per relazione binaria tra entità non principale ed entità principale

**Esempio 6.6.** Un esempio di applicazione è il seguente:

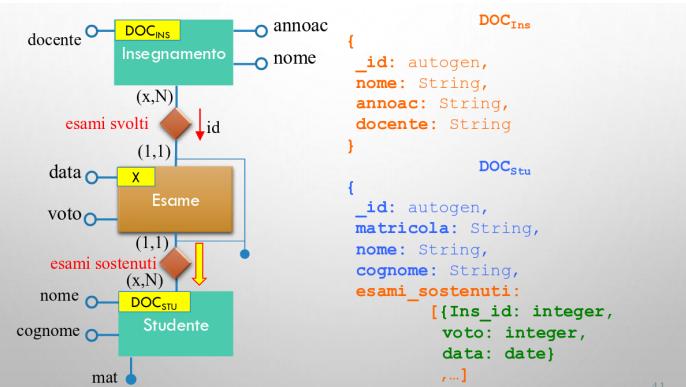


Figura 79: Esempio di traduzione di relazione binaria tra entità non principale ed entità principale

- ii. Relazione binaria  $P_2$  tra un'entità non principale  $E_i$ , da incapsulare in  $EP_1$ , e un'entità principale  $EP_2$  con vincolo  $(X, N)$  lato  $E_i$ :

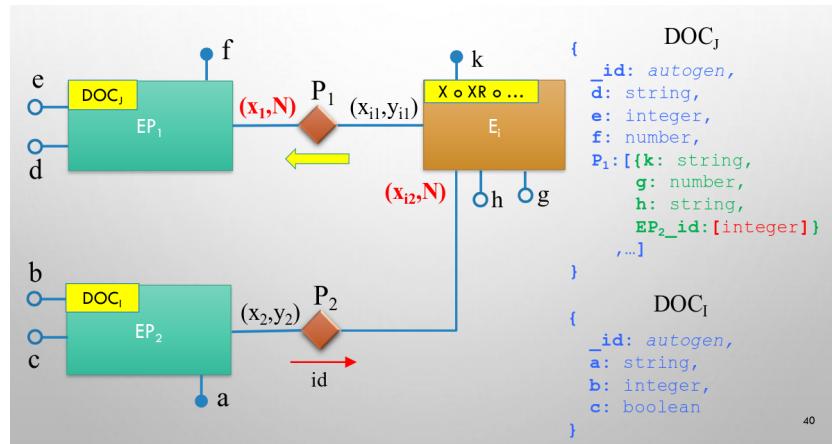


Figura 80: Regola di traduzione per relazione binaria tra entità non principale ed entità principale

(g) Regola 7:

- Relazione binaria  $P_2$  tra due entità non principali  $E_i, E_j$ , da encapsulare in  $EP_1$  ed  $EP_2$  rispettivamente, con vincolo  $(X, 1)$  lato  $E_i$ :

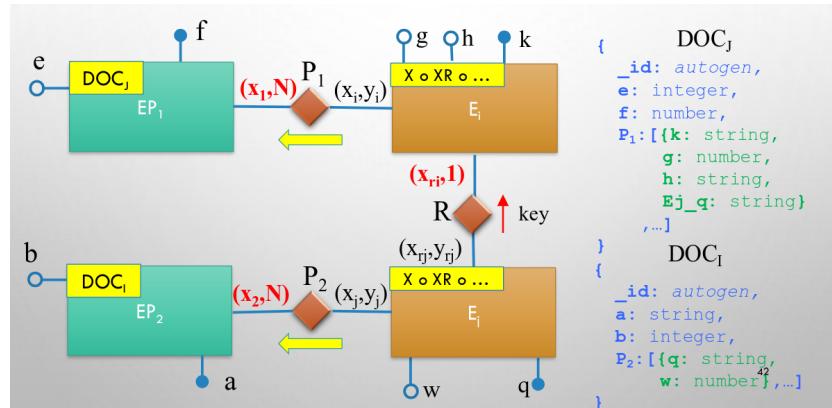


Figura 81: Regola di traduzione per relazione binaria tra due entità non principali

- Relazione binaria  $P_2$  tra due entità non principali  $E_i, E_j$ , da encapsulare in  $EP_1$  ed  $EP_2$  rispettivamente, con vincolo  $(X, N)$  lato  $E_i$ :

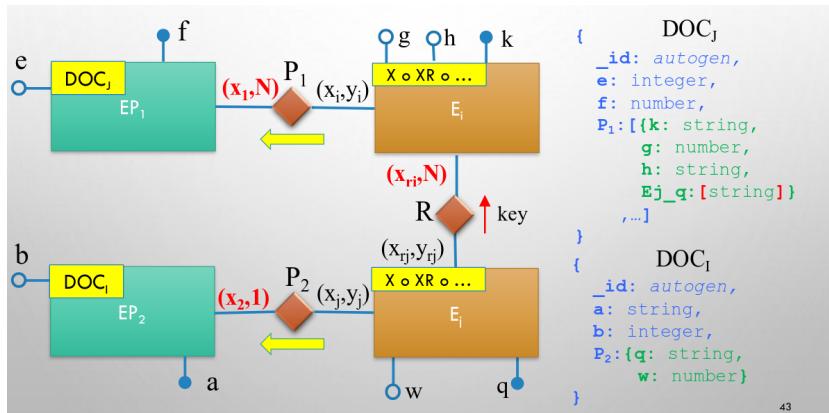


Figura 82: Regola di traduzione per relazione binaria tra due entità non principali

## 7 Operazioni su una base di dati relazionale

I linguaggi per gestire i dati sono divisi in due categorie a seconda del tipo di operazioni che consentono di eseguire:

- **Data Definition Language (DDL)**: è usato per la **definizione** dei dati. Questo linguaggio consente di:
  - Creare lo schema della base di dati
  - Modificare lo schema della base di dati
- **Data Manipulation Language (DML)**: è usato per la **manipolazione** dei dati. Questo linguaggio consente di:
  - Inserire o cancellare tuple nelle relazioni
  - Aggiornare i valori nelle tuple delle relazioni
  - **Interrogare le relazioni** per estrarre informazioni

I linguaggi di interrogazione che fanno parte del DML si dividono in due categorie:

- **Linguaggi procedurali**: specificano il **procedimento** per ottenere il risultato. Ad esempio l'algebra relazionale.
- **Linguaggi dichiarativi**: specificano le **proprietà** del risultato. Ad esempio il calcolo relazionale e SQL (Structured Query Language).

### 7.1 Algebra relazionale

L'algebra relazionale è un linguaggio procedurale per l'interrogazione delle basi di dati relazionali. Questo linguaggio è un insieme di **operatori** su **relazioni**.

### 7.1.1 Operatori dell'algebra relazionale

La "segnatura" degli operatori dell'algebra relazionale è la seguente:

- Operatore **unario**:

$$op_{\text{parametri}}(r_1) \rightarrow r_2$$

Dove  $r_1$  e  $r_2$  sono relazioni e  $op$  è l'operatore con i suoi parametri specifici.

- Operatore **binario**:

$$r_1 op_{\text{parametri}} r_2 \rightarrow r_3$$

Dove  $r_1$ ,  $r_2$  e  $r_3$  sono relazioni e  $op$  è l'operatore con i suoi parametri specifici.

Gli operatori si classificano in due modi:

- Classificazione **funzionale**: distingue
  - operatori **insiemistici**
  - operatori **specifici**
  - operatori di **giunzione (o join)** tra relazioni
- Classificazione in base alla **derivabilità**: distingue
  - operatori **di base**
  - operatori **derivati**

### 7.1.2 Operatori insiemistici

Siccome le relazioni sono **insiemi di tuple** ha senso applicare ad esse le operazioni insiemistiche. Tuttavia va considerato che le relazioni sono insiemi di **tuple omogenee**, cioè tutte le tuple di una relazione hanno lo stesso **schema** (stesso insieme di attributi con stessi domini). Quindi le operazioni insiemistiche si possono applicare **soltanto a relazioni con lo stesso schema**.

Date due relazioni  $r_1$  e  $r_2$  sullo **stesso schema X** si definiscono i seguenti operatori.

- **Operatori di base**:

- **Unione**:

$$r_1 \cup r_2 = r_3 = \{t \mid t \in r_1 \vee t \in r_2\}$$

schema: X

**Nota:** tutte le tuple duplicate vengono rappresentate una sola volta.

- **Differenza**:

$$r_1 - r_2 = r_3 = \{t \mid t \in r_1 \wedge t \notin r_2\}$$

schema: X

**Nota:** la differenza non è commutativa.

- **Operatori derivati**:

– **Intersezione:**

$$r_1 \cap r_2 = r_1 - (r_1 - r_2)$$

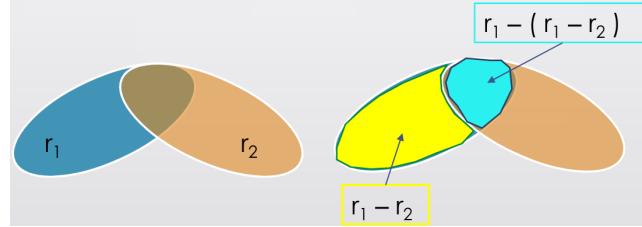


Figura 83: Operatore di intersezione

La cardinalità degli operatori insiemistici è il numero di tuple contenute nella relazione risultante:

- Unione:

$$\max(|r_1|, |r_2|) \leq |r_1 \cup r_2| \leq |r_1| + |r_2|$$

- Differenza:

$$0 \leq |r_1 - r_2| \leq |r_1|$$

- Intersezione:

$$0 \leq |r_1 \cap r_2| \leq \min(|r_1|, |r_2|)$$

### 7.1.3 Operatori specifici

Data una relazione  $r_1$  sullo schema  $X = \{A_1, A_2, \dots, A_n\}$ , si definiscono i seguenti operatori (tutti unari):

- **Operatori di base:**

- **Ridenominazione:** cambia il nome degli attributi di una relazione. Sia  $Y = \{B_1, B_2, \dots, B_n\}$  un insieme di attributi con  $|Y| = |X|$ , allora:

$$\rho_{A_1, \dots, A_n \rightarrow B_1, \dots, B_n}(r_1) = r_2 = \{t \mid \exists t' \in r_1 : \forall i \in [1, n]. t[B_i] = t'[A_i]\}$$

schema:  $Y$

**Esempio 7.1.** Consideriamo lo schema relazionale:

CAPOLUOGO_PROV(Nome, Popolazione, CAP)
COMUNI_MINORI(Nome, Abitanti, CAP)

Si richiede di produrre la lista di tutti i comuni:

$$\rho_{\text{Popolazione} \rightarrow \text{Abitanti}}(\text{CAPOLUOGO\_PROV}) \\ \cup \text{COMUNI\_MINORI}$$

- **Selezione:** consente di estrarre da una relazione le tuple che soddisfano una certa condizione  $F(t)$ :

$$\sigma_F(r_1) = r_2 = \{t \mid t \in r_1 \wedge F(t)\}$$

schema: X

La condizione  $F(t)$  è una formula proposizionale con le seguenti caratteristiche:

- \* Si ottiene combinando con i connettivi logici  $\wedge, \vee$  e  $\neg$  condizioni atomiche del tipo:  $A \Theta B$  o  $A \Theta c$ , dove:
  - $A, B \in X$
  - $\Theta \in \{<, =, >, \neq, \geq, \leq\}$
  - $c \in \text{DOM}(A)$  è una costante compatibile con il dominio di A
- \* Una formula  $A \Theta B$  è vera sulla tupla  $t$  se:

$$t[A] \Theta t[B]$$

- \* Una formula  $A \Theta c$  è vera sulla tupla  $t$  se:

$$t[A] \Theta c$$

- \* Le formule  $F_1 \wedge F_2$ ,  $F_1 \vee F_2$  e  $\neg F$  hanno valori booleani definiti dalle note tabelle di verità dei connettivi logici corrispondenti.

**Esempio 7.2.** Consideriamo lo schema relazionale:

$$\text{TRENO}(\underline{\text{Numero}}, \text{PartOra}, \text{PartMin}, \text{Cat}, \text{Dest}, \\ \text{ArrOra}, \text{ArrMin})$$

$$\text{FERMATA}(\underline{\text{NumTreno}}, \underline{\text{Stazione}}, \text{Ora}, \text{Min})$$

Si richiede di produrre la lista di tutti i treni non regionali che partono alle 12.00 o dopo le 12.00 e prima delle 16.00.

$$\sigma_{\text{PartOra} \geq 12 \wedge \text{PartOra} < 16 \wedge \text{Cat} \neq \text{'Regionale'}}(\text{TRENO})$$

- **Proiezione:** riduce le tuple di una relazione in verticale, cioè elimina attributi. Sia  $Y = \{A_1, \dots, A_m\} \subseteq X$  un sottoinsieme degli attributi di  $X$  allora:

$$\Pi_Y(r_1) = r_2 = \{t \mid \exists t' \in r_1 : t'[Y] = t\}$$

schema: Y

dove  $t'[Y]$  indica una tupla  $t_0$  su Y tale che:

$$\forall A_i \in Y : t_0[A_i] = t'[A_i]$$

**Esempio 7.3.** Consideriamo lo schema relazionale:

TRENO(Numero, PartOra, PartMin, Cat, Dest,  
ArrOra, ArrMin)

FERMATA(NumTreno, Stazione, Ora, Min)

Si richiede di restituire il numero e l'ora di partenza dei treni Freccia Rossa (categoria = 'FR') con destinazione 'Milano Centrale'.

$\Pi_{\text{Numero}, \text{PartOra}}(\sigma_{\text{Cat}=\text{FR} \wedge \text{Dest}=\text{'Milano Centrale'}}(\text{TRENO}))$

La cardinalità degli operatori specifici è il numero di tuple contenuto nella relazione risultante:

- Selezione:

$$0 \leq |\sigma_F(r_1)| \leq |r_1|$$

- Proiezione:

$$\min(|r_1|, 1) \leq |\Pi_Y(r_1)| \leq |r_1|$$

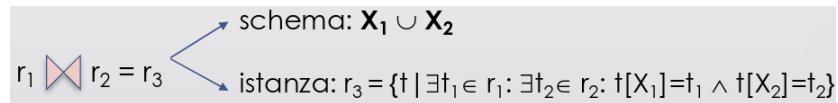
$|\Pi_Y(r_1)| = |r_1|$  se  $Y$  è superchiave per  $r_1$ .

#### 7.1.4 Operatori di giunzione

Date due relazioni  $r_1$  e  $r_2$  con schemi  $X_1$  e  $X_2$  rispettivamente, gli operatori di **join** generano tuple  $t$  nella relazione risultante a partire dalle coppie di tuple  $(t_1, t_2) \in r_1 \times r_2$  che soddisfano una certa condizione chiamata **predicato di join**.

- Operatori di base:

- **Join naturale**: il predicato di join in questo operatore si basa sugli attributi comuni alle due relazioni (è dipendente dallo schema).



Il predicato di join è隐式的 (implicito).

#### Proprietà:

- \* Il join naturale si dice **completo** se tutte le tuple di  $r_1$  e  $r_2$  contribuiscono a generare almeno una tupla della relazione risultante:

$$\begin{aligned} \forall t_1 \in r_1 : \exists t \in r_1 \bowtie r_2 : t[X_1] = t_1 \\ \wedge \\ \forall t_2 \in r_2 : \exists t \in r_1 \bowtie r_2 : t[X_2] = t_2 \end{aligned}$$

- \* Se il join naturale non è completo le tuple che non contribuiscono al risultato si dicono **dangling tuples**.

- \* Il join naturale è **commutativo**:

$$r_1 \bowtie r_2 = r_2 \bowtie r_1$$

- \* Il join naturale è **associativo**:

$$r_1 \bowtie (r_2 \bowtie r_3) = (r_1 \bowtie r_2) \bowtie r_3$$

- \* Se  $r_1$  e  $r_2$  hanno lo stesso schema, allora:

$$r_1 \bowtie r_2 = r_1 \cap r_2$$

- \* Se  $r_1$  e  $r_2$  non hanno attributi in comune, allora:

$$r_1 \bowtie r_2 = r_1 \times r_2 \quad \text{Tutte le possibili coppie}$$

**Esempio 7.4.** Consideriamo gli schemi relazionali:

DOCENTE(CFDocente, Nome, Cognome)  
CORSO(Nome, CFDocente)

Si richiede di produrre l'insieme dei corsi riportando: nome del corso e cognome del docente

$$\begin{aligned} & \Pi_{\text{NomeCorso}, \text{Cognome}}( \\ & \rho_{\text{Nome} \rightarrow \text{NomeCorso}}(\text{CORSO}) \bowtie \text{DOCENTE} \\ & ) \end{aligned}$$

#### • Operatori derivati:

- **Theta join**: il predicato di join in questo operatore viene indicato esplicitamente (**è indipendente dallo schema**). C’è la precondizione che gli schemi di  $r_1$  e  $r_2$  **devono essere disgiungi**:

$$X_1 \cap X_2 = \emptyset$$

Il theta join si rappresenta come:

$$r_1 \bowtie_{\Theta} r_2 = \sigma_{\Theta}(r_1 \bowtie r_2)$$

#### Proprietà:

- \* Il theta join si dice **equi-join** se la condizione  $\Theta$  è una congiunzione di uguaglianze:

$$r_1 \bowtie_{A_1=B_1 \wedge \dots \wedge A_n=B_n} r_2$$

**Nota bene:** non esiste un operatore misto che applica la logica del join naturale per gli attributi comuni e aggiunge la condizione specificata esplicitamente attraverso il predicato  $\Theta$ .

#### Proprietà dell’equi-join:

- Date due relazioni  $r_1$  e  $r_2$  di schema  $X_1$  e  $X_2$  con  $X_1 \cap X_2 = C_1, \dots, C_m$  con  $m > 0$ , vale la seguente equivalenza:

$$r_1 \bowtie r_2 = \Pi_{X_1 \cup X_2} ( r_1 \bowtie_{C_1=C'_1 \wedge \dots \wedge C_m=C'_m} \rho_{C_1, \dots, C_m \rightarrow C'_1, \dots, C'_m} (r_2) )$$

**Forme brevi:** Data la relazione  $r_1$  di schema  $X_1 = \{A_1, \dots, A_n\}$  per cambiare nome a tutti gli attributi si può scrivere:

$$\rho_{A_1, \dots, A_n \rightarrow A'_1, \dots, A'_n} (r_1)$$

oppure in forma breve:

$$\rho_{X \rightarrow X'} (r_1)$$

**Nota bene:** non è una notazione ammessa dall'algebra relazionale la dot notation per rinominare gli attributi, cioè non è ammesso scrivere:

$$r_1.A_1 \rightarrow r_1.A'_1$$

**Esempio 7.5.** Consideriamo gli schemi relazionali:

DOCENTE(CF, Nome, Cognome)

CORSO(Nome, Docente)

CORSO.Docente  $\rightarrow$  DOCENTE.CF

Si richiede di produrre l'elenco di tutti i corsi riportando: nome del corso e cognome del docente.

$$\begin{aligned} &\Pi_{\text{NomeCorso}, \text{Cognome}} ( \\ &\quad \rho_{\text{Nome} \rightarrow \text{NomeCorso}} (\text{CORSO}) \bowtie_{\text{CF}=\text{Docente}} \text{DOCENTE} \\ &\quad ) \end{aligned}$$

La cardinalità del join naturale è:

- In generale:

$$0 \leq |r_1 \bowtie r_2| \leq |r_1| \cdot |r_2|$$

- Se il join è **completo**:

$$\max(|r_1|, |r_2|) \leq |r_1 \bowtie r_2| \leq |r_1| \cdot |r_2|$$

- Se  $X_1 \cap X_2$  è una superchiave per  $r_2$ :

$$0 \leq |r_1 \bowtie r_2| \leq |r_1|$$

- Se  $X_1 \cap X_2$  è una superchiave per  $r_2$  ed esiste un vincolo di integrità referenziale tra  $X_1 \cap X_2$  (o una parte) di  $r_1$  e  $r_2$ :

$$|r_1 \bowtie r_2| = |r_1|$$

La cardinalità del join è sempre quella dove si esporta la chiave.

## 7.2 Algebra con valori nulli

In una base di dati relazionale, un attributo può assumere il valore speciale **null** che indica l'assenza di un valore noto. È quindi opportuno estendere l'algebra relazionale per fare in modo che possa manipolare anche relazioni che contengono valori nulli. Le operazioni che devono essere estese per gestire relazioni con valori nulli sono:

- **Selezione**

Le condizioni di selezione in presenza di valori nulli hanno i seguenti valori di verità:

- $A \Theta B$  sulla tupla  $t$ : se  $t[A]$  o  $t[B]$  sono null, allora  $t[A] \Theta t[B]$  è **falso**:
- $A \Theta \text{cost}$  sulla tupla  $t$ : se  $t[A]$  è null, allora  $t[A] \Theta \text{cost}$  è **falso**
- Condizioni atomiche aggiuntive:  $\text{AISNULL}$  e  $\text{AISNOTNULL}$ :
  - \*  $\text{AISNULL}$  validato sulla tupla  $t$  è **vero** se  $t[A]$  contiene null, altrimenti è **falso**.
  - \*  $\text{AISNOTNULL}$  validato sulla tupla  $t$  è **vero** se  $t[A]$  non contiene null, altrimenti è **falso**.

- **Join naturale**

La condizione di uguaglianza sugli attributi comuni alle due relazioni è falsa sulle tuple  $t_1$  e  $t_2$  se **almeno uno degli attributi comuni** di  $t_1$  o  $t_2$  è **null**.

### 7.2.1 Join esterni

(Questi operatori non sono ammessi in questo corso).

Consentono di ottenere nel risultato del join tutte le tuple (anche le tuple pendenti "dangling tuples") di una o di entrambe le relazioni coinvolte nel join, eventualmente estese con valori nulli:

- **Left join**:  $r_1 \bowtie_{\text{LEFT}} r_2$
- **Right join**:  $r_1 \bowtie_{\text{RIGHT}} r_2$
- **Full join**:  $r_1 \bowtie_{\text{FULL}} r_2$

### 7.2.2 Ottimizzazione di espressioni DML

Ogni espressione DML (solitamente specificata in **linguaggio dichiarativo**) ricevuta dal DBMS è soggetta ad un processo di **elaborazione**.

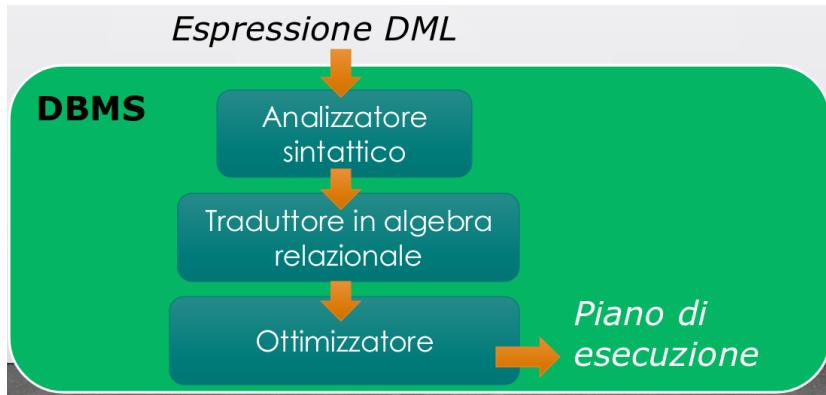


Figura 84: Ottimizzazione di espressioni DML

L'**ottimizzatore** genera un'espressione equivalente all'interrogazione in input e di costo inferiore. Il costo viene valutato in termini di **dimensione dei risultati intermedi**. L'ottimizzatore esegue **trasformazioni di equivalenza** allo scopo di **ridurre la dimensione dei risultati intermedi**.

### 7.2.3 Trasformazioni di equivalenza

Esistono diversi tipi di equivalenze tra espressioni algebriche:

- **Equivalenza dipendente dallo schema**

Dato uno schema R:

$$E_1 \equiv_R E_2 \text{ se } E_1(r) = E_2(r) \text{ per ogni istanza } r \text{ di schema } R$$

ad esempio:

$$\Pi_{AB}(R_1) \bowtie \Pi_{AC}(R_2) \equiv_R \Pi_{ABC}(R_1 \bowtie R_2)$$

con  $R = \{R_1(A, B, D), R_2(A, C, E)\}$

- **Equivalenza assoluta**

È indipendente dallo schema:

$$E_1 \equiv E_2 \text{ se } E_1 \equiv_R E_2 \text{ per ogni schema } R \text{ compatibile con } E_1 \text{ e } E_2$$

ad esempio:

$$\Pi_{AB}(\sigma_{A>0}(R_1)) \equiv \sigma_{A>0}(\Pi_{AB}(R_1))$$

Considerando E un'espressione di schema X, si definiscono le seguenti trasformazioni di equivalenza:

- **Atomizzazione delle selezioni:**

$$\sigma_{F_1 \wedge F_2}(E) \xrightarrow{\quad} \sigma_{F_1}(\sigma_{F_2}(E))$$

È propedeutica ad altre trasformazioni. Non ottimizza se non è seguita da altre trasformazioni.

- **Idempotenza delle proiezioni:**

$$\Pi_Y(E) \xrightarrow{\equiv} \Pi_Y(\Pi_{YZ}(E)) \text{ dove } Z \subseteq X$$

È propedeutica ad altre trasformazioni. Non ottimizza se non è seguita da altre trasformazioni.

Siano  $E_1$  e  $E_2$  espressioni di schema  $X_1$  e  $X_2$ , si definiscono le seguenti trasformazioni di equivalenza:

- **Anticipazione delle selezioni rispetto al join:**

$$\sigma_F(E_1 \bowtie E_2) \xrightarrow{\equiv} E_1 \bowtie \sigma_F(E_2)$$

Applicabile solo se  $F$  si riferisce solo ad attributi di  $E_2$ .

- **Anticipazione della proiezione rispetto al join:**

$$\Pi_{X_1 Y}(E_1 \bowtie E_2) \xrightarrow{\equiv} E_1 \bowtie \Pi_Y(E_2)$$

Applicabile solo se  $Y \subseteq X_2$  e  $(X_2 - Y) \cap X_1 = \emptyset$

Combinando l'anticipazione della proiezione con l'idempotenza delle proiezioni otteniamo:

$$\Pi_Y(E_1 \bowtie_F E_2) \xrightarrow{\equiv} \Pi_Y(\Pi_{Y_1}(E_1) \bowtie_F \Pi_{Y_2}(E_2))$$

$$\Pi_Y(E_1 \bowtie E_2) \xrightarrow{\equiv} \Pi_Y(\Pi_{Y_1}(E_1) \bowtie \Pi_{Y_2}(E_2))$$

dove:

- $Y_1 = (X_1 \cap Y) \cup J_1$
- $Y_2 = (X_2 \cap Y) \cup J_2$
- $J_1(J_2)$  sono gli attributi di  $E_1(E_2)$  coinvolti nel join (vale a dire presenti in  $F$  per il theta-join, mentre in caso di join naturale  $J_1 = J_2 = X_1 \cap X_2$ )

### 7.3 Calcolo relazionale

È un linguaggio di interrogazione dichiarativo, cioè specifica le proprietà del risultato dell'interrogazione. Esistono due versioni del calcolo relazionale:

- Il calcolo relazionale sui domini
- Il calcolo relazionale sulle tuple (è il fondamento teorico del linguaggio SQL)

(in questo corso si vede solo il calcolo relazionale sulle tuple).

### 7.3.1 Caratteristiche del linguaggio

Nel calcolo relazionale sulle tuple l'interrogazione viene specificata attraverso **una formula logica con variabili libere**, dove la formula viene interpretata sul **contenuto della base di dati** e le variabili assumono come valori le tuple delle relazioni della base di dati.

Il calcolo relazionale sulle tuple definisce la semantica del linguaggio SQL "semplice" (senza operatori aggregati e senza join esterni) includendo le interrogazioni nidificate.

**Definizione 7.1.** Una variabile è **libera** in una formula logica  $F$ , se non compare in un quantificatore (esistenziale  $\exists$  o universale  $\forall$ ) di  $F$ .

**Esempio 7.6.** Alcuni esempi di variabili libere sono:

1.  $F_1(x) \equiv x > 3$
2.  $F_2(x, y) \equiv x > 3 \wedge x < 6 \wedge y = 10$
3.  $F_3(x) \equiv y > 0 \wedge \exists y(y < 0)$
4.  $F_4(x) \equiv x > 3 \wedge \exists y(y < x)$
5.  $F_5(z) \equiv z < 10 \wedge \forall y(y < z)$
6.  $F_6(y) \equiv y < 10 \vee \forall x(y > x)$

L'interpretazione è la scelta degli insiemi da cui prelevare i valori da sostituire nelle variabili. Ad esempio, si può scegliere di interpretare le formule nell'insieme degli interi  $\mathbb{Z}$  o nei naturali  $\mathbb{N}$ , ecc...

### 7.3.2 Sintassi del calcolo relazionale sulle tuple

Le espressioni del calcolo relazionale sulle tuple sono della forma:

$$\{T|L|F\}$$

dove:

- $T$  è la **target list**, cioè definisce lo **schema della relazione risultato** e come le tuple risultato si ottengono dalle variabili libere.  $T$  è una lista di elementi separati da virgole:

$$e_1, \dots, e_n$$

dove ogni elemento  $e_i$  può essere:

- $Y : x.(Z)$ , dove  $Y$  e  $Z$  sono sequenze di attributi con la stessa cardinalità e  $x$  è una variabile libera.
- $x.(Z)$ , dove  $Z$  è una sequenza di attributi e  $x$  è una variabile libera (equivale a  $Z : x.(Z)$ ).

- $x.*$  dove  $x$  è una variabile libera; in questo caso gli attributi sono esattamente tutti quelli della relazione associata alla variabile  $x$  nella range list  $L$ .
  - $L$  è la **range list**, cioè definisce le **variabili libere** e le **collega** alle relazioni della base di dati coinvolte nell'interrogazione.
- $L$  è una lista di elementi separati da virgolet:

$$r_1, \dots, r_m$$

dove ogni elemento  $r_i$  è della seguente forma:

$$x_j(R)$$

dove  $x_j$  è una variabile libera e  $R$  è il nome di una relazione dello schema della base di dati. Esiste **uno e un solo** elemento  $r_i$  nella range list per ogni variabile libera della formula  $F$ .

- $F$  è una **formula** che specifica **la condizione che deve essere soddisfatta** dalle variabili libere, cioè dalle tuple coinvolte nell'interrogazione.

$F$  può essere:

– **Formula atomica:**

$$* x.A \Theta c \text{ oppure } x_1.A_1 \Theta x_2.A_2$$

dove  $x, x_1, x_2$  sono variabili libere,  $A, A_1, A_2$  sono attributi,  $c$  è una costante e  $\Theta \in \{=, \neq, >, <, \geq, \leq\}$ .

– **Formula non atomica:**

\* se  $f_1$  e  $f_2$  sono formule, allora anche  $f_1 \wedge f_2$ ,  $f_1 \vee f_2$ ,  $\neg f_1$  e  $\neg f_2$  lo sono.

\* se  $f$  è una formula, allora anche  $\forall x(R)(f)$  e  $\exists x(R)(f)$  lo sono.

Nota bene:

- Nelle formule atomiche tutte le variabili sono **libere**.
- Nelle formule:

$$\forall x(f) \text{ e } \exists x(f)$$

$x$  è una variabile **legata**, mentre tutte le altre variabili sono libere o legate se sono libere o legate in  $f$ .

- Le congiunzioni, disgiunzioni e la negazione non cambiano l'insieme delle variabili libere o legate di una formula.

### 7.3.3 Semantica del calcolo relazionale sulle tuple

Le formule vengono interpretate sull'istanza corrente della base di dati:

$$db = \{r_1, \dots, r_n\}$$

di schema:

$$S = \{R_1(X_1), \dots, R_n(X_n)\}$$

Ogni formula contiene un certo numero di variabili libere:

$$f(x_1, \dots, x_n)$$

Data una ennupla di tuple  $(t_1, \dots, t_n)$ , tale ennupla rende vera la formula se, quando si sostituiscono le tuple  $t_1, \dots, t_n$  alle variabili libere  $x_1, \dots, x_n$ , tale sostituzione soddisfa la formula.

- **Formule atomiche:**

- $x_1.A_1 \Theta x_2.A_2$  è vera sulle tuple  $(t_1, t_2)$  se il confronto:

$$t_1[A_1] \Theta t_2[A_2]$$

è soddisfatto.

- $x.A \Theta c$  è vera sulla tupla  $t_1$  se il confronto:

$$t_1[A] \Theta c$$

è soddisfatto.

- **Formule non atomiche:**

- Le formule  $f_1 \wedge f_2$ ,  $f_1 \vee f_2$ ,  $\neg f_1$  e  $\neg f_2$  sono vere secondo le usuali definizioni dei connettivi logici (tabelle di verità).

- **Formule con quantificatori:**

- $\exists x(R_i)(f)$  con variabili libere  $(x_1, \dots, x_q)$  è vera sulle tuple  $(t_1, \dots, t_q)$ , se **esiste almeno** una tupla  $t \in r_i$  (istanza di  $R_i$ ) tale che  $f$  è vera quando si sostituiscono le tuple  $(t, t_1, \dots, t_q)$  alle variabili  $(x, x_1, \dots, x_q)$ .
- $\forall x(R_i)(f)$  con variabili libere  $(x_1, \dots, x_q)$  è vera sulle tuple  $(t_1, \dots, t_q)$ , se **per ogni** tupla  $t \in r_i$  (istanza di  $R_i$ ),  $f$  è vera quando si sostituiscono le tuple  $(t, t_1, \dots, t_q)$  alle variabili  $(x, x_1, \dots, x_q)$ .

#### 7.3.4 Semantica di una interrogazione

L'interpretazione di una espressione del calcolo come interrogazione:

$$Q = \{Y_1 : x_1.(Z_1), \dots, Y_k : x_k.(Z_k) \mid x_1(R_1), \dots, x_n(R_n) \mid f(x_1, \dots, x_n)\}$$

La valutazione dell'interrogazione  $Q$  produce una relazione risultato  $R$  dove:

- Lo schema di  $R$  è:

$$Y = Y_1 \cup \dots \cup Y_k$$

- $R$  contiene tutte le tuple  $t$  su  $Y$  tali che esiste una ennupla di tuple  $(t_1, \dots, t_n) \in R_1 \times \dots \times R_n$  che generano  $t$  e che sostituite alle variabili libere  $(x_1, \dots, x_n)$  rendono vera  $f$ .

#### Osservazioni:

- Il calcolo relazionale sulle tuple **non è equivalente all'algebra relazionale** e **non è equivalente al calcolo relazionale sui domini** in quanto l'unione di due relazioni non è rappresentabile nel calcolo relazionale sulle tuple.

- È invece possibile rappresentare nel calcolo relazionale sulle tuple sia l'intersezione che la differenza tra due relazioni.

L'espressione del calcolo  $\{T \mid L \mid F\}$  è in diretta corrispondenza con la forma base delle interrogazioni SQL:



**Esempio 7.7.** Consideriamo lo schema relazionale:

TRENO(Num, Cat, Part, Arr, Dest)  
FERMATA(Treno, Staz, Orario)

Si richiede di restituire il numero e l'ora di partenza dei treni Freccia Rossa (Cat = 'FR') con destinazione 'Milano Centrale'.

$\{\text{Numero, Orario}_p \text{artenza} : x.(\text{Num, Part}) \mid$   
 $x(\text{TRENO}) \mid$   
 $x.\text{Cat} = 'FR' \wedge x.\text{Dest} = 'MilanoCentrale'\}$

### 7.3.5 Operatori insiemistici nel calcolo relazionale sulle tuple

- **Intersezione:** L'intersezione tra due relazioni  $R_1$  e  $R_2$  di shema  $X = \{A_1, \dots, A_n\}$  si esprime come:

$$R_1 \cap R_2 = \{x.* \mid x(R_1), y(R_2) \mid x.A_1 = y.A_1 \wedge \dots \wedge x.A_n = y.A_n\}$$

oppure:

$$R_1 \cap R_2 = \{x.* \mid x(R_1) \mid \exists y(R_2)(x.A_1 = y.A_1 \wedge \dots \wedge x.A_n = y.A_n)\}$$

- **Differenza:** La differenza tra due relazioni  $R_1$  e  $R_2$  di shema  $X = \{A_1, \dots, A_n\}$  si esprime come:

$$R_1 - R_2 = \{x.* \mid x(R_1) \mid \nexists y(R_2)(x.A_1 = y.A_1 \wedge \dots \wedge x.A_n = y.A_n)\}$$

- **Unione:** L'unione non è esprimibile in quanto non è possibile associare una variabile libera a più relazioni della base di dati.

### 7.3.6 Semantica dei quantificatori

Le espressioni che possono essere specificate attraverso i quantificatori ( $\exists, \forall$ ) corrispondono in SQL a clausole WHERE nelle quali compaiono **predicati complessi che richiedono interrogazioni nidificate**.

- $\exists x(R_i)(f)$  con variabili libere  $(x_1, \dots, x_q)$  è vera sulle tuple  $(t_1, \dots, t_q)$ , se esiste almeno una tupla  $t \in r_i$  tale che  $f$  è vera quando si sostituiscono le tuple  $(t, t_1, \dots, t_q)$  alle variabili  $(x, x_1, \dots, x_q)$ .
- $\forall x(R_i)(f)$  con variabili libere  $(x_1, \dots, x_q)$  è vera sulle tuple  $(t_1, \dots, t_q)$ , se per ogni tupla  $t \in r_i$ ,  $f$  è vera quando si sostituiscono le tuple  $(t, t_1, \dots, t_q)$  alle variabili  $(x, x_1, \dots, x_q)$ .

Il quantificatore esistenziale si può eliminare **solo quando non è negato**.

**Esempio 7.8.** Consideriamo lo schema relazionale:

TRENO(Num, Cat, Part, Arr, Dest)  
 FERMATA(Treno, Staz, Orario)

Trovare il numero e la categoria dei treni che fermano a 'Vicenza' e fermano a 'Padova' dopo le 20.30.

```
{
    Numero, Categoria : x.(Num, Cat) |
    x(TRENO) |
    ∃y(FERMATA)(x.Num = y.Treno ∧ y.Staz = 'Vicenza') ∧
    ∃z(FERMATA)(
        x.Num = z.Treno ∧ z.Staz = 'Padova' ∧ z.Orario > '20:30'
    )
}
```

oppure:

```
{
    Numero, Categoria : x.(Num, Cat) |
    x(TRENO), y(FERMATA), z(FERMATA) |
    x.Num = y.Treno ∧ y.Staz = 'Vicenza' ∧
    x.Num = z.Treno ∧ z.Staz = 'Padova' ∧ z.Orario > '20:30'
}
```

## 7.4 Interrogazioni nei sistemi document based

Nei sistemi NoSQL spesso non sono presenti veri e propri linguaggi di interrogazione come nei sistemi relazionali. Solo i sistemi document based forniscono strumenti simili ad algebra relazionale e calcolo relazionale per l'accesso ai dati. I due principali sistemi document based sono:

- Couchbase: Fornisce un linguaggio SQL++, simile all'SQL e dichiarativo
- MongoDB: Fornisce una serie di operatori che consentono di scrivere complesse espressioni algebriche (pipeline) per estrarre dati dalle collezioni di documenti

Buona parte del linguaggio di interrogazione di MongoDB è realizzato attraverso il metodo find.

**Esempio 7.9.** Considerando la seguente basi di dati di seguito ci sono alcuni esempi di interrogazioni in MongoDB.

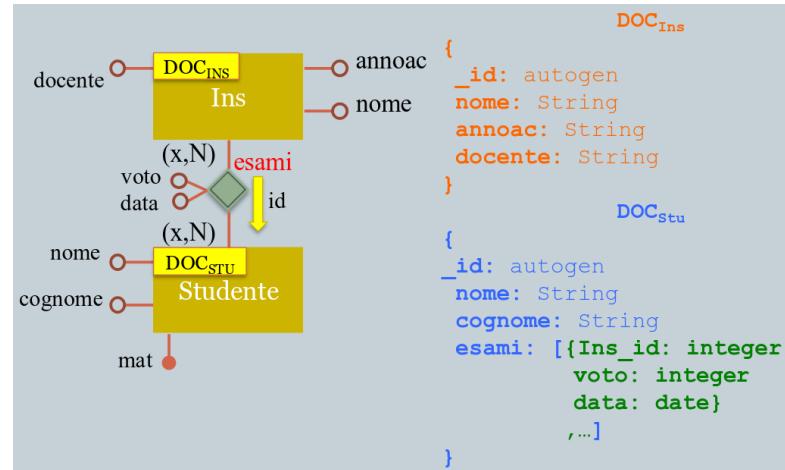


Figura 85: Esempio di interrogazione in MongoDB

Un esempio di inserimenti è il seguente:

```

1 db.studenti.insertMany([
2     { _id: "VR00010",
3         nome: "Mario",
4         cognome: "Rossi",
5         esami: [
6             {
7                 ins: "Basi di dati",
8                 voto: 22,
9                 data: "1/7/2016"
10            },
11            {
12                ins: "Reti di calcolatori",
13                voto: 28,
14                data: "15/9/2016"
15            }
16        ]
17    },
18    { _id: "VR00011",
19        nome: "Luigi",
20        cognome: "Verdi",
21        esami: [
22            {
23                ins: "Basi di dati",
24                voto: 30,
25                data: "10/7/2016"
26            },
27            {
28                ins: "Reti di calcolatori",
29

```

```

30         voto: 26,
31         data: "20/9/2016"
32     }
33   ]
34 })
35 ])

```

Alcune interrogazioni semplici sono:

- Trovare tutti gli studenti:

```
1 db.studenti.find({})
```

- Trovare tutti gli studenti con cognome 'Rossi':

```
1 db.studenti.find({ Cognome: "Rossi" })
```

- Trovare tutti gli studenti di cognome 'Rossi' e nome 'Mario':

```
1 db.studenti.find({ Cognome: "Rossi", Nome: "Mario" })
```

- Trovare tutti gli studenti di cognome 'Rossi' o 'Verdi':

```
1 db.studenti.find({
2   Cognome: { $in: [ "Rossi", "Verdi" ] }
3 })
```

Tutti gli operatori hanno il prefisso \$. In questo caso l'operatore \$in consente di specificare una condizione di appartenenza ad un insieme.

#### 7.4.1 Operatori logici e di confronto in MongoDB

Gli operatori logici e di confronto in MongoDB sono:

- **Operatori logici:**

- \$and: congiunzione logica
- \$or: disgiunzione logica
- \$not: negazione logica
- \$nor: negazione della disgiunzione logica

- **Operatori di confronto:**

- \$eq: uguale
- \$ne: diverso
- \$gt: maggiore di
- \$gte: maggiore o uguale
- \$lt: minore di
- \$lte: minore o uguale
- \$in: appartiene ad un insieme
- \$nin: non appartiene ad un insieme

**Esempio 7.10.** Alcuni esempi di utilizzo di questi operatori sono:

- { \$or: [  
2     { Cognome: "Rossi" },  
3     { Cognome: "Verdi" }  
4 ]  
5 }
  
- { \$and: [  
2     { Cognome: "Rossi" },  
3     { Nome: "Verdi" }  
4 ]  
5 }
  
- { \$and: [  
2     { Esami.voto: { \$gte: 21 } },  
3     { Esami.voto: { \$lte: 26 } }  
4 ]  
5 }

#### 7.4.2 Interrogazione di dati incapsulati

Per interrogare dati incapsulati in array o documenti annidati si usa il punto (.) come operatore di accesso ai campi.

**Esempio 7.11.** Alcuni esempi di interrogazioni di dati incapsulati sono:

- Trovare tutti gli studenti che hanno registrato **almeno un esame** con voto 30:

```
1 db.studenti.find({ Esami.voto: 30 })
```

- Trovare tutti gli studenti che hanno registrato **almeno un esame** con voto maggiore di 22:

```
1 db.studenti.find({ Esami.voto: { $gt: 22 } })
```

#### 7.4.3 Interrogazioni con proiezione

La proiezione in MongoDB si ottiene specificando un secondo argomento al metodo find.

**Esempio 7.12.** Alcuni esempi di interrogazioni con proiezione sono:

- Trovare il cognome degli studenti di nome 'Mario':

```
1 db.studenti.find(  
2     { Nome: "Mario" },  
3     { Cognome: 1 }  
4 )
```

- Trovare il cognome degli studenti di nome 'Mario' escludendo il campo `_id`:

```

1 db.studenti.find(
2   { Nome: "Mario" },
3   { Cognome: 1, _id: 0 }
4 )

```

- Trovare la matricola e i voti di tutti gli studenti:

```

1 db.studenti.find(
2   {},
3   { _id: 1, "Esami.voto": 1 }
4 )

```

#### 7.4.4 Interrogazioni con join

Per eseguire il join tra due collezioni in MongoDB bisogna usare l'operazione **aggregate** con l'operatore \$lookup. Questo tipo di query è sconsigliato dal sistema MongoDB in quanto meno efficiente.

**Esempio 7.13.** Considerando il seguente schema di base di dati:

```

STUDENTI: {_id: "VR00010",
            nome: "Mario",
            cognome: "Rossi" }
CORSI: {_id: "ins01",
         nome: "Basi di dati",
         annoac: "2016/2017",
         docente: "Belussi" }
{_id: "ins02",
  nome: "Algebra",
  annoac: "2015/2016",
  docente: "Gregorio"}
ESAMI: {_id: "01",
        stud_id: "VR00010"
        ins_id: "ins01",
        voto: 22,
        data: "1/7/2016" }
{_id: "02", stud_id: "VR00010",
  ins_id: "ins02",
  voto: 26,
  data: "5/7/2015" }

```

Figura 86: Schema di base di dati in MongoDB per join

Un esempio di interrogazione con join è:

```

db.STUDENTI.aggregate([
  { $lookup:
    { from: ESAMI,
      localField: _id,
      foreignField: stud_id,
      as: "esami_fatti"
    }
  }
])

```

Figura 87: Esempio di join in MongoDB

Il risultato di questa interrogazione è:

```

{_id: "VR00010",
  nome: "Mario",
  cognome: "Rossi",
  "esami_fatti": [
    {_id: "01", stud_id: "VR00010",
      ins_id: "ins01",
      voto: 22,
      data: "1/7/2016" },
    {_id: "02", stud_id: "VR00010",
      ins_id: "ins02",
      voto: 26,
      data: "5/7/2015" }
  ]
}

```

Figura 88: Risultato di join in MongoDB

#### 7.4.5 Interrogazioni su dati encapsulati con condizioni multiple

Per eseguire interrogazioni su dati encapsulati con condizioni multiple in MongoDB si usa l'operatore \$elemMatch.

**Esempio 7.14.** Un esempio di interrogazione su dati encapsulati con condizioni multiple è:

- Trovare tutti gli studenti che hanno ottenuto un voto maggiore di 22 nell'appello del 30/10/2025:

```

1 db.studenti.find({
2   esami: {
3     $elemMatch: {
4       data: "30/10/2025",
5       voto: { $gt: 22 }
6     }
7   }
}

```

8 } )