



DIPARTIMENTO  
DI INFORMATICA  
**SAPIENZA**  
UNIVERSITÀ DI ROMA

---

# Basi di Dati II

---

*Authors*

Emanuele D'Agostino  
Giuseppe Borracci

# Indice

---

## 0 Introduzione

0.1 Ciclo di vita del software a cascata

    0.1.1 Analisi dei requisiti

0.2 Unified Modelling Language (UML)

    0.2.1 Limiti dell'UML

## 1 Il linguaggio Entity-Relationship

1.1 Introduzione al linguaggio ER

    1.1.1 Livello intensionale vs livello estensionale

    1.1.2 Costrutti del linguaggio ER

1.2 Entità

    1.2.1 Attributo di entità

    1.2.2 Rappresentazione di entità e attributi

    1.2.3 Domini degli attributi

1.3

    1.3.1

1.4

    1.4.1

1.5

    1.5.1

# 0 Introduzione

---

## 0.1 Ciclo di vita a cascata di un software

Un software può essere realizzato seguendo uno tra molteplici schemi di sviluppo e mantenimento. Nel caso del ciclo di vita a cascata, ci basiamo su una serie di passaggi eseguiti sequenzialmente:

### 1. Studio di fattibilità e raccolta dei requisiti

- Valutare costi e benefici
- Pianificare le attività e le risorse del progetto
- Individuare l'ambiente di programmazione (hardware/software)
- Raccogliere i requisiti

### 2. Analisi dei requisiti

Si occupa del cosa l'applicazione dovrà realizzare.

- Descrivere il dominio dell'applicazione e specificare le funzioni delle varie componenti nello schema concettuale

### 3. Progetto e realizzazione

Si occupa del come l'applicazione dovrà realizzare le sue funzioni.

- Definire l'architettura del programma
- Scegliere le strutture di rappresentazione
- Scrivere il codice del programma e produrre la documentazione

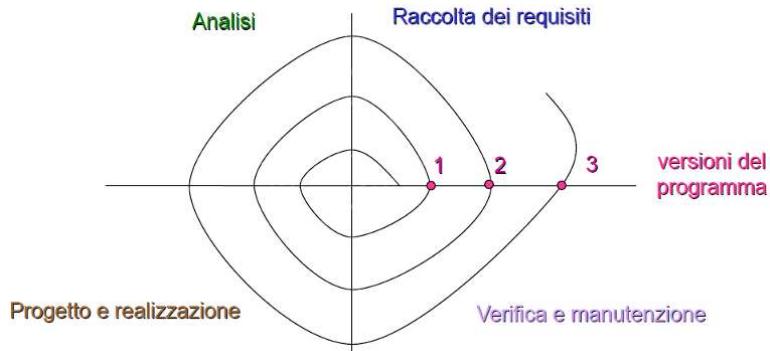
### 4. Verifica

- Il programma svolge correttamente, completamente, efficientemente il compito per cui è stato sviluppato?

### 5. Manutenzione

- Controllo del programma durante l'esercizio
- Correzione e aggiornamento del programma

Al termine di ogni fase, se necessario, si può tornare ad una fase precedente (in tal caso si parla di modello a spirale).



### 0.1.1 Analisi dei requisiti

La fase di analisi dei requisiti nel ciclo di sviluppo del software, è caratterizzata da:

**Input** Requisiti raccolti

**Output** Schema concettuale dell'applicazione

L'obiettivo è di costruire un modello dell'applicazione che sia:

- Completo
- Preciso
- Leggibile
- Traducibile in un programma eseguibile

A cosa serve:

- **Analizzare i requisiti:**
  - ▶ Coglie le loro implicazioni
  - ▶ Li specifica con l'obiettivo di formalizzarli e di eliminare incompletezze, inconsistenze e ambiguità
- **Creare un modello** (schema concettuale) che sarà un riferimento per tutte le fasi successive del ciclo di vita del software
- **Verificare i requisiti** con l'utente finale
- Prendere decisioni fondamentali sulla strutturazione e sulla modularizzazione del software
- **Fornire la specifica** delle funzionalità da realizzare

Lo schema concettuale è composto da **diagrammi** (in opportuni linguaggi grafici di modellazione) e **documenti di specifica**, i quali descrivono completamente e precisamente il sistema da realizzare secondo diverse prospettive, come ad esempio:

- Quali sono e come sono organizzati i dati di interesse
- Quali sono le funzionalità da offrire e a quali utenti
- Come evolvono i dati di interesse nel tempo

NB:

In questa fase ci si concentra su **cosa** e non su **come** (indipendenza da aspetti realizzativi/tecnologici).

## 0.2 Unified Modelling Language (UML)

UML fornisce costrutti per modellare gran parte degli aspetti, sia statici che dinamici (dati, funzioni, evoluzione), usando un approccio Object Oriented.

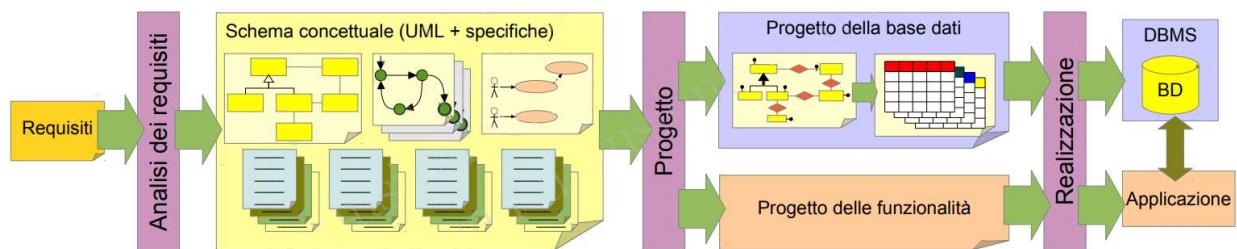
Si tratta di un linguaggio estremamente vasto (ed anche estendibile), che permette di produrre diagrammi per descrivere:

- L'organizzazione degli oggetti (dati) di interesse in gerarchie di classi (diagrammi delle classi e degli oggetti).

- L'evoluzione permessa da singoli oggetti nel tempo (diagrammi degli stati e transizioni).
- Le funzionalità offerte dal sistema e i relativi attori (utenti umani o sistemi esterni) che possono utilizzarle (diagrammi degli use case).
- Le interazioni tra oggetti e processi del sistema (diagrammi di sequenza, di collaborazione, delle attività).
- L'architettura del sistema (diagrammi dei componenti, diagrammi di deployment).

Un approccio unificante alla progettazione del software include la progettazione di una base dati:

- la fase di **Analisi** viene condotta usando UML e documenti aggiuntivi di specifica per modellare il sistema sotto le diverse prospettive, quindi anche per decidere quali siano i dati di interesse e per comprenderne la struttura
- nella **fase di Progetto** si decide come implementare la memorizzazione di tali dati, ad esempio usando un DBMS
- l'utilizzo di una base dati realizzata mediante un DBMS può quindi essere vista come una scelta progettuale, basata su considerazioni tecnologiche, e non una scelta di Analisi.



### 0.2.1 Limiti dell'UML

In UML, l'analisi degli aspetti relativi ai dati confluisce nel diagramma delle classi, che però contiene anche informazioni circa alcune funzioni (operazioni sui dati).

Alcuni vincoli sui dati non sono esprimibili in modo succinto in UML, che è un linguaggio molto generale.

Nella fase di Analisi, si continuano quindi ad usare linguaggi di modellazione pre-UML esplicitamente orientati a modellare dati (e non funzioni) che, nell'approccio unificato, sono previsti solo in fase di Progetto.

Il più diffuso linguaggio di modellazione concettuale dei dati è Entity-Relationship (ER).

# 1 Il linguaggio Entity-Relationship

---

## 1.1 Introduzione al linguaggio ER

Il linguaggio ER permette di creare un **modello dei dati** di interesse per un'applicazione (utilizzato in fase di **Analisi dei requisiti**).

ER è un **linguaggio grafico** che fornisce dei **costrutti** (elementi di diagrammi ER) che vanno usati rispettando una sintassi ed a cui sono associate una semantica e sintassi precise per descrivere formalmente una formula logica.

### 1.1.1 Livello intensionale vs livello estensionale

Un diagramma ER descrive la struttura dei dati, non i dati (che possono variare).

Si dice che ER descrive il livello intensionale dei dati:

- Livello intensionale (struttura)
- Livello estensionale (istanze)

Esempi:

- *classi* (aspetto intensionale) vs *oggetti* (aspetto estensionale) in un linguaggio object-oriented
- *struct* (aspetto intensionale) vs *istanze di struct* (aspetto estensionale) in C

Ad ogni diagramma ER (livello intensionale) corrispondono in genere più livelli estensionali (insiemi di istanze), anche se in ogni momento, solo uno è quello significativo (quello che rappresenta lo stato corrente del mondo rappresentato).

### 1.1.2 Costrutti del linguaggio ER

Tali costrutti sono proprio gli elementi utilizzati per costruire un diagramma ER. Tra cui distinguiamo:

- Entità
- Relationship
- Attributi (di entità o relationship)
- Vincoli di integrità
- Relazioni is-a tra entità e tra relationship
- Generalizzazione tra entità

## 1.2 Entità

Un'entità rappresenta una classe di oggetti (fatti, persone, cose) di interesse per il dominio applicativo. Le istanze di un'entità hanno proprietà comuni e, ciascuna istanza, esiste indipendentemente dalle altre.

Esempi di entità:

- Persona
- Azienda
- Impiegato
- Fattura

In ogni momento, ciascuna entità rappresenta un insieme di istanze (astrazione).

### 1.2.1 Attributo di entità

È una proprietà locale di un'entità di interesse per il dominio applicativo che associa ad ogni istanza di entità un valore in un certo dominio (tipo).

Esempi:

- Attributi per l'entità "Impiegato" → nome (stringa), età, stipendio (interi positivi)
- Attributi per l'entità "Azienda" → nome (stringa), sede (stringa)

Il valore di un attributo di una istanza di entità dipende solo dall'istanza stessa, non ha (in genere) alcun rapporto con le altre istanze.

### 1.2.2 Rappresentazione di entità e attributi

Prendiamo come esempio due entità: Impiegato e Azienda (per ora guardiamole come fossero classi).

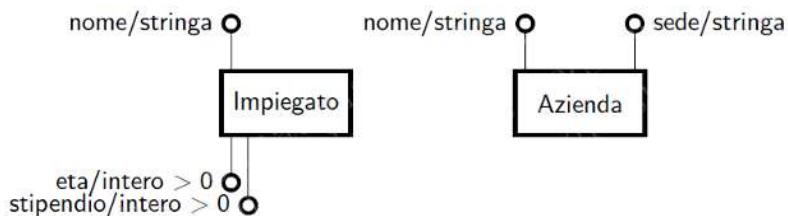
Ogni istanza di entità Impiegato ha (in questo esempio) associato:

- Uno ed un solo valore di tipo stringa per l'attributo nome
- Uno ed un solo valore di tipo intero  $> 0$  per l'attributo età
- Uno ed un solo valore di tipo intero  $> 0$  per l'attributo stipendio

Ogni istanza di entità Azienda ha associato:

- Uno ed un solo valore di tipo stringa per l'attributo nome
- Uno ed un solo valore di tipo stringa per l'attributo sede

Rappresenteremo le due entità nel seguente modo:



Siccome abbiamo detto che **ogni istanza esiste indipendentemente dalle altre**, possono quindi coesistere due istanze con valori uguali, poiché restano comunque due istanze diverse.

Ad esempio possiamo avere due istanze diverse del tipo:

- nome = "Anna"      età = 35      stipendio = 40000
- nome = "Anna"      età = 35      stipendio = 40000

Questo accade in quanto possono esistere due persone diverse che hanno però lo stesso nome, la stessa età e percepiscono lo stesso stipendio (ma ciò non le rende due persone uguali).

### 1.2.3 Domini degli attributi

Possiamo assumere che siano disponibili vari tipi elementari come domini di attributi:

- |           |            |        |
|-----------|------------|--------|
| ► Stringa | ► Reale    | ► Data |
| ► Intero  | ► Booleano | ► Ora  |

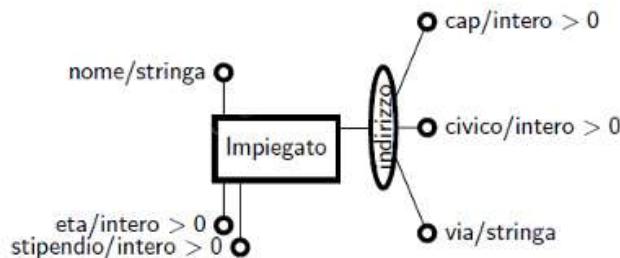
Per rendere la modellazione più aderente alla realtà, possiamo restringere i domini suddetti mediante **vincoli di dominio**:

- Definendo un limite, come ad esempio "intero  $> 0$ ", "intero  $\geq 0$ " o "reale  $< 0$ ".
- O con la notazione  $[x, y]$  con cui poniamo il dominio nell'intervallo di interi tra  $x$  e  $y$ .

Un altro dominio che possiamo utilizzare liberamente nei diagrammi ER per l'Analisi è il **dominio enumerativo**, espresso come insieme di simboli dati esplicitamente, ad esempio  $\{\text{uomo, donna}\}$ .

### 1.2.4 Attributi composti

Possiamo inoltre specificare un attributo su un dominio di tipo record, avente campi su domini base o (a loro volta) record **attributo composto**.

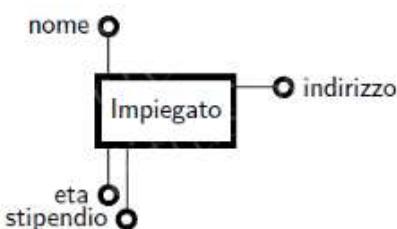


Per semplicità, i domini degli attributi non vengono indicati nel diagramma ER. Essi compaiono in un documento allegato.

Domini per gli attributi dell'entità Impiegato:

Attributo	Dominio
nome	stringa
età	intero $> 0$
stipendio	intero $> 0$
indirizzo	record(via: stringa, civico: intero $> 0$ , cap: intero $> 0$ )

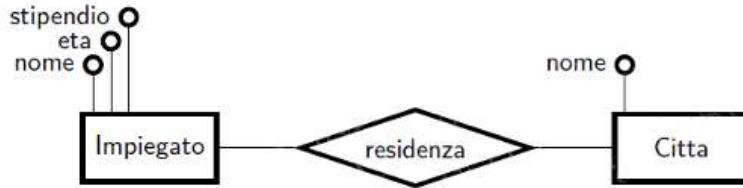
Anche i campi di un attributo composto possono essere omessi e indicati in un documento allegato.



## 1.3 Relationship

Una relationship esprime la possibilità di legami tra istanze di due o più entità. Il numero di entità coinvolte in una relationship si chiama **grado** o **arità** della relationship.

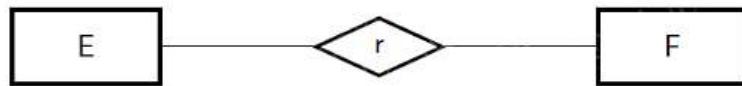
Poniamo ad esempio il caso di voler conoscere la città di residenza degli impiegati, esprimeremo la relationship come:



Il livello estensionale di una relationship  $r$  tra le entità  $E$  ed  $F$  è costituito da un insieme di coppie  $(e, f)$  tali che  $e$  è un'istanza di  $E$  ed  $f$  è un'istanza di  $F$ . Se consideriamo  $E$  ed  $F$  come gli insiemi delle loro rispettive istanze, abbiamo:

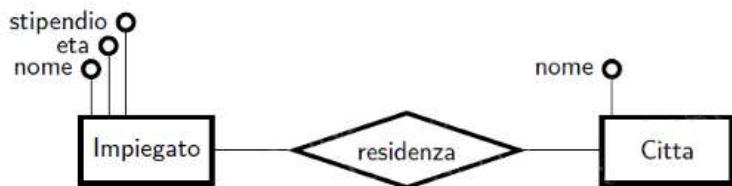
$$r \subseteq E \times F$$

- ovvero  $r$  è un sottoinsieme del prodotto cartesiano  $E \times F$
- ovvero  $r$  è una relazione matematica su  $E$  ed  $F$



Notiamo quindi che  $r$  in quanto insieme, è una collezione di valori senza ripetizioni. Non possono quindi esistere in  $r$  due istanze che legano la stessa coppia di istanze di  $E$  e di  $F$ .

Esempio 1:



Supponiamo che, a livello estensionale:

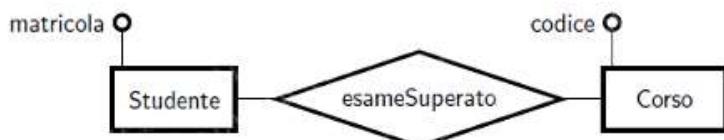
$$\text{Impiegato} = \{\text{anna}, \text{mario}\} \quad \text{Città} = \{\text{milano}, \text{roma}\}$$

Prendiamo come esempio due possibili livelli estensionali per residenza:

$(\text{anna}, \text{milano})$	$(\text{anna}, \text{milano})$
$(\text{mario}, \text{roma})$	$(\text{mario}, \text{roma})$
$(\text{anna}, \text{milano}) \leftarrow$ duplicato, non ammesso!	

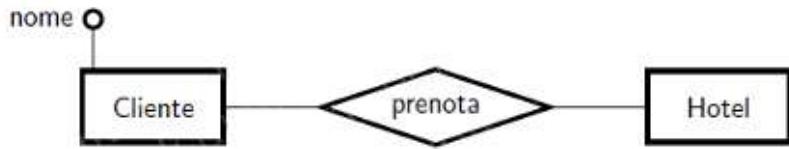
Non avrebbe infatti senso rappresentare due volte che l'impiegato di nome Anna risiede a Milano.

Esempio 2:



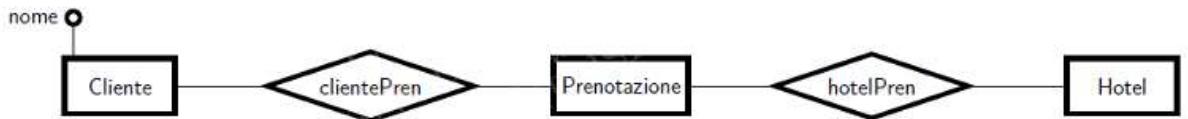
Grazie alla restrizione imposta dal costrutto **relationship**, stiamo affermando che *"uno stesso studente non può superare più volte lo stesso esame"*.

Esempio 3:



In questo caso, l'uso della relationship impedisce livelli estensionali che vorremmo ammettere, in quanto, in questo modo, *uno stesso cliente non può prenotare più volte lo stesso hotel*.

Se vogliamo distinguere le diverse prenotazioni di ogni cliente ad ogni hotel, dobbiamo usare il concetto di entità:



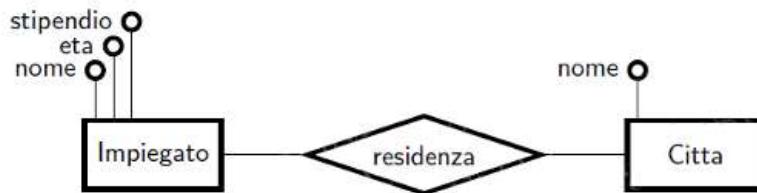
<i>mario</i>	<i>(mario, p1)</i>	<i>p1</i>	<i>(p1, hotel1)</i>	<i>hotel1</i>
<i>anna</i>	<i>(anna, p2)</i>	<i>p2</i>	<i>(p2, hotel2)</i>	<i>hotel2</i>
	<i>(mario, p3)</i>	<i>p3</i>	<i>(p3, hotel1)</i>	

In questo modo, ogni singola prenotazione è modellata come un'istanza di entità, e quindi ha vita propria, indipendentemente dalla coppia cliente/hotel *p1* e *p3* sono oggetti distinti.

### 1.3.1 Vincoli di integrità

I diagrammi ER visti fino ad ora sono modelli molto laschi della realtà di interesse.

Prendiamo ad esempio la relationship vista prima:



È ammesso un possibile livello estensionale che non dovrebbe esistere:

<i>mario</i>	<i>(mario, roma)</i>	<i>roma</i>
<i>mario</i>	<i>(mario, milano)</i>	<i>milano</i>

In tal caso stiamo affermando che Mario ha residenza sia a Roma che a Milano, vorremmo però che tale relazione sia univoca (ogni persona può risiedere solo in uno luogo in un dato momento).

A questo scopo usiamo quindi i vincoli di integrità, i quali sono quindi regole (espresse sul diagramma) che impongono restrizioni ai livelli estensionali ammessi.

Esistono diverse tipologie di vincoli di integrità, la prima tipologia di vincoli di integrità che vedremo esprime restrizioni sul **numero** di volte in cui un'istanza di entità può essere coinvolta in una relationship. Questi vincoli vanno sotto il nome di vincoli di **cardinalità** sulle relationship e li rappresentiamo nel seguente modo:



In questo esempio stiamo esprimendo i seguenti vincoli:

- Quante istanze di relationship (coppie impiegato/città) possono coinvolgere lo stesso impiegato?

**Da 1 ad 1 ⇒ esattamente 1**

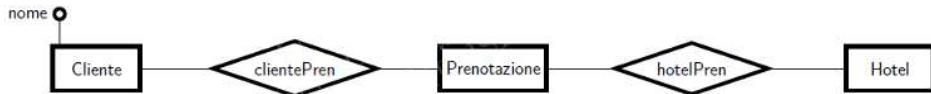
Ogni impiegato ha una ed una sola città di residenza.

- Quante istanze di relationship (coppie impiegato/città) possono coinvolgere la stessa città?

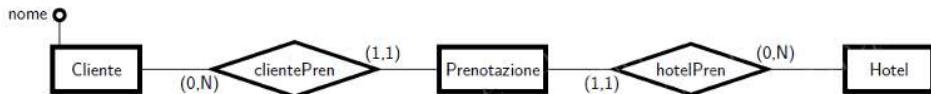
**Da 0 ad N ⇒ nessun limite**

Ogni città può essere residenza di un numero arbitrario (anche 0) di impiegati.

Torniamo al diagramma visto che modella clienti, hotel e prenotazioni:



Aggiungiamo adesso gli adeguati vincoli di cardinalità sulle due relationship:

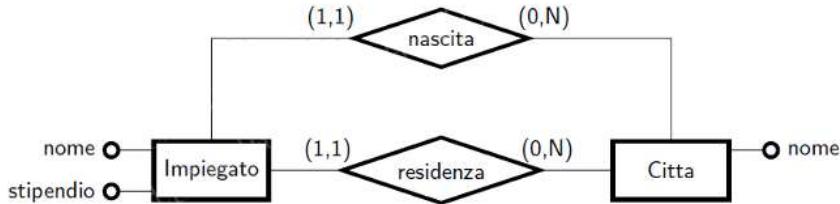


- Ogni istanza di Prenotazione deve essere coinvolta in:
  - esattamente una istanza di *clientePren* (perché è relativa ad *un* cliente)
  - esattamente una istanza di *hotelPren* (perché è relativa ad *un* hotel)
- Ogni istanza di Cliente può essere coinvolta in un numero arbitrario di istanze di *clientePren* (ogni cliente può effettuare un numero *arbitrario* di prenotazioni)
- Ogni istanza di Hotel può essere coinvolta in un numero arbitrario di istanze di *hotelPren* (ogni hotel può ricevere un numero *arbitrario* di prenotazioni)

### 1.3.2 Entità coinvolte in più relationship

Analizzeremo adesso il caso in cui due o più entità sono coinvolte in più di una relationship.

Esempio 1:



Le relationship residenza e nascita esprimono legami di tipo diverso.

Esempio 2:

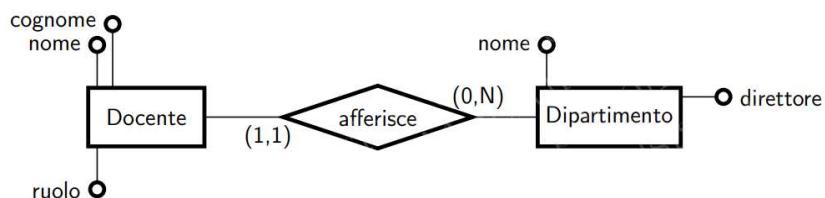
Analizziamo adesso un problema più pratico.

Vogliamo modellare i docenti di un ateneo, di ogni docente interessa:

- nome e cognome
- ruolo che può ricoprire: RU (ricercatore universitario), PA (prof. associato), PO (prof. ordinario)
- dipartimento

Dei dipartimenti interessa il nome ed il direttore.

Soluzione 1:



Entità Docente

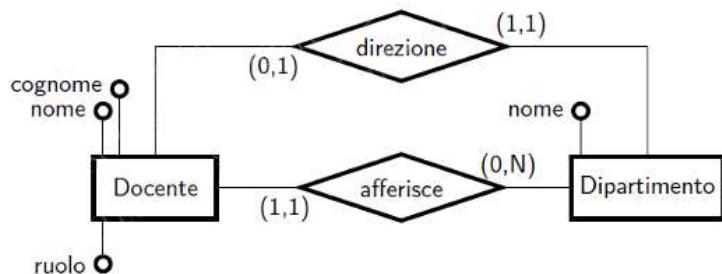
Entità Dipartimento

attributo	dominio	attributo	dominio
nome	stringa	nome	stringa
cognome	stringa	direttore	stringa
ruolo	{RU, PO, PA}		

**ERRORE!**

In questo modo si è modellato il direttore di un dipartimento come una stringa, mentre in realtà è un'istanza dell'entità Docente.

Soluzione 2:

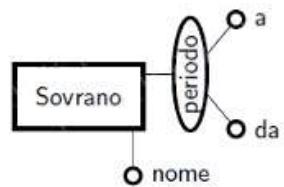


Entità Docente		Entità Dipartimento	
attributo	dominio	attributo	dominio
nome	stringa	nome	stringa
cognome	stringa		
ruolo	{RU, PO, PA}		

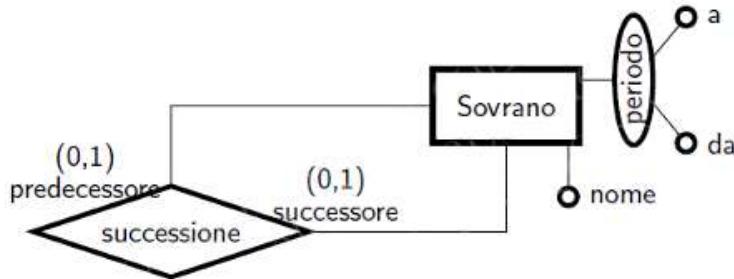
Notiamo infatti che "*un docente o non dirige niente o dirige al più un solo dipartimento (0,1)*" e che "*ogni dipartimento è gestito da uno e un solo direttore (1, 1)*".

### 1.3.3 Relationship che coinvolgono più volte un'entità

Supponiamo di voler modellare i sovrani di un regno, dove di ogni sovrano interessa il nome, il periodo in cui ha regnato, ed il predecessore.



Modelliamo ora il concetto di predecessore



Le istanze della relationship *successione* sono coppie  $(p_1, p_2) : p_1, p_2 \in \text{Sovrano}$ .

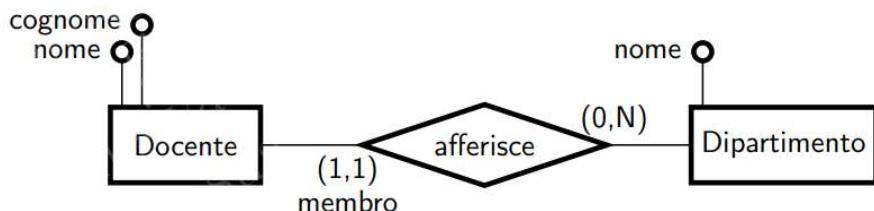
Per evitare ambiguità, diamo un nome ai ruoli che le istanze di entità Sovrano giocano nei legami (istanze di relationship).

Una istanza di relationship *successione* diventa una coppia etichettata:

$$(\text{predecessore: } p_1, \quad \text{successore: } p_2)$$

### 1.3.4 Ruoli delle entità nelle relationship

Per ogni entità  $E$  coinvolta in una relationship  $r$ , è possibile specificare i ruoli di  $E$  in  $r$ , sugli archi che collegano  $E$  ad  $r$ . I nomi dei ruoli per una relationship  $r$  devono essere distinti. La specifica dei ruoli è obbligatoria per una relationship che insiste più volte su una entità.

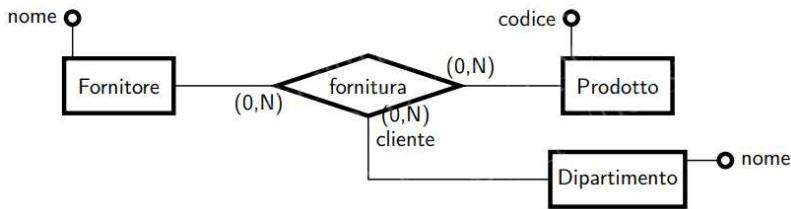


Se il ruolo di un'entità in una relationship non è indicato, si assume che abbia nome uguale a quello dell'entità. In questo caso il ruolo di Dipartimento in *afferisce* è: *Dipartimento*.

### 1.3.5 Relationship con aritÀ (grado) maggiore di 2

Una relationship può legare anche più di due entità.

Esempio (relationship ternaria):



A livello estensionale, la relationship fornitura rappresenta un insieme di terne etichettate:

$$(Fornitore:f, cliente:d, Prodotto:p)$$

Tali che

$$f \in Fornitore, d \in Dipartimento, p \in Prodotto.$$

La semantica dei vincoli di cardinalità è analoga al caso di relationship tra due entità.

### 1.3.6 Semantica delle relationship (versione finale)

Siamo quindi pronti a dare la semantica completa di una relationship tra un numero arbitrario (almeno 2) di entità, non necessariamente tutte distinte.

A livello estensionale, una relationship  $r$  tra le entità  $E_1, E_2, \dots, E_n$  (non necessariamente tutte distinte), con ruoli, rispettivamente  $u_1, u_2, \dots, u_n$  è costituita da un insieme di  $n$ -ple etichettate della forma

$$(u_1: x_1, u_2: x_2, \dots, u_n: x_n)$$

tali che

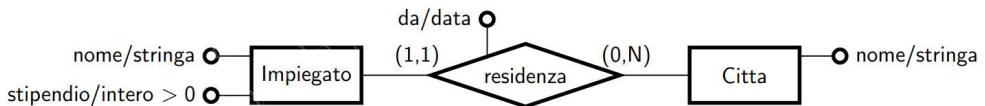
$$x_1 \in E_1, x_2 \in E_2, \dots, x_n \in E_n .$$

## 1.4 Attributi di relationship

Un attributo di relationship è una proprietà locale di una relationship che associa ad ogni istanza di relationship (ennupla di istanze di entità) un valore in un certo dominio (tipo).

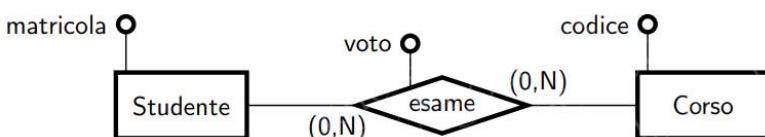
Il valore di un attributo di una istanza di relationship dipende solo dall'istanza stessa, non ha (in genere) alcun rapporto con le altre istanze .

Esempio:



Ad ogni istanza  $(i, c) \in residenza$  (con  $i \in Impiegato$  e  $c \in Città$ ) è associato un valore per l'attributo  $da$  sul dominio  $data$ .

Esempio:



Entità *Studente*

Attributo Dominio  
matricola intero

Entità *Esame*

Attributo Dominio  
voto [18, 30]

Entità *CORSO*

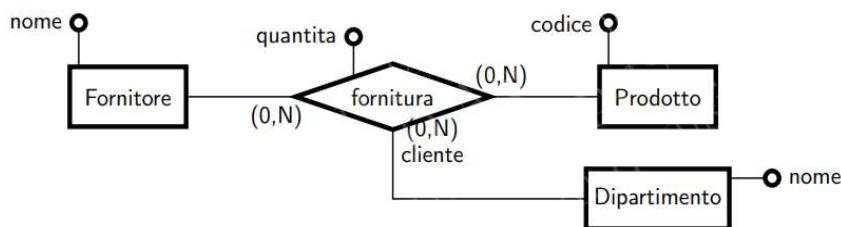
Attributo Dominio  
codice intero

NB:

Un'istanza di relationship *esame* è ancora una coppia  $(s, c)$  con  $s \in \text{Studente}$  e  $c \in \text{CORSO}$ , alla quale è però associato un valore  $v \in [18, 30]$ . Un esame non potrà quindi essere dato più di una volta dallo stesso studente (si avrebbero due *n-uple* uguali). In particolare, un'istanza di *esame* non è una terna  $(p, c, v)$ ! continuano a non poter coesistere due istanze di *esame* che legano la stessa coppia *studente/corso* (anche se con voti diversi)

Anche relationship di arità maggiore di 2 possono avere attributi.

Esempio:



A livello estensionale, la relationship *fornitura* rappresenta un *insieme di terne etichettate*:

$(\text{Fornitore}: f, \text{cliente}: d, \text{Prodotto}: p)$

tali che  $f \in \text{Fornitore}$ ,  $d \in \text{Dipartimento}$ ,  $p \in \text{Prodotto}$ , ad ognuna delle quali è associato un valore per l'attributo *quantità*.

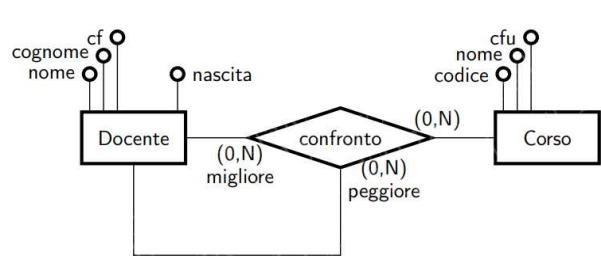
Non possono inoltre coesistere due terne uguali (indipendentemente dal valore per l'attributo).

Esempio:

Si vogliono modellare i seguenti requisiti per un sistema informativo universitario. Sono di interesse per l'applicazione i docenti ed i corsi.

Dei docenti si vuole rappresentare nome, cognome, codice fiscale e data di nascita, dei corsi si vuole mantenere codice identificativo, nome e numero di crediti.

Sfruttando i moduli di valutazione dei corsi e dei docenti da parte degli studenti, si vuole poi rappresentare l'informazione circa quale docente è più apprezzato di quale altro come insegnante di un corso.



Entità *Docente*

Attributo	Dominio
nome	stringa
cognome	stringa
cf	stringa
nascita	data

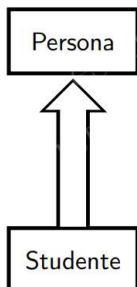
Entità *CORSO*

Attributo	Dominio
codice	intero
nome	stringa
cfu	intero > 0

## 1.5 Relazioni IS-A tra entità

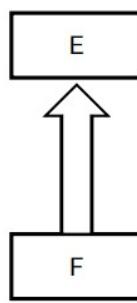
Fino ad ora abbiamo implicitamente assunto che entità diverse non hanno istanze in comune. In molte situazioni, vogliamo rappresentare il fatto che tra due entità sussista una relazione di *sottoinsieme*.

Il diagramma ER permette di definire il concetto di relazione *is-a* tra entità.



Ogni istanza di *Studente* è anche (is-a) un'istanza di *Persona*.

- ▶ *Persona* è l'*entità base*
- ▶ *Studente* è l'*entità derivata* (o entità figlia, o sotto-entità)

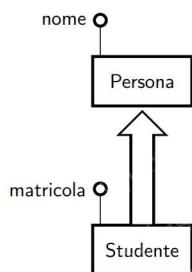


A livello estensionale, in ogni momento, l'insieme delle istanze di *F* deve essere un sottoinsieme (anche non proprio) dell'insieme delle istanze di *E*

Esempio:

- ▶ Istanze di *F* = {*a, c*}
- ▶ Istanze di *E* = {*a, b, c, d*}

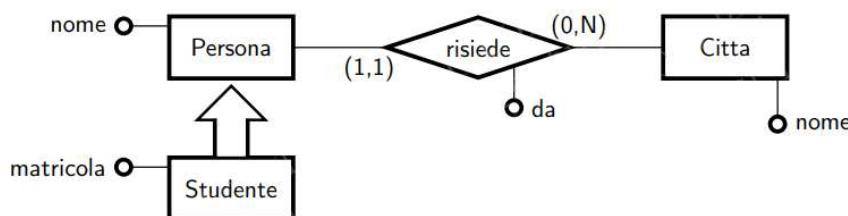
### 1.5.1 Ereditarietà su entità: attributi



Nell'esempio di fianco ogni istanza di *Persona* ha un valore per l'attributo *nome*. Ogni istanza di *Studente* è anche (is-a) istanza di *Persona*, perciò ogni istanza di *Studente* ha un valore per l'attributo *nome*.

L'entità *Studente* può avere ulteriori attributi (nell'esempio: *matricola*) non appartenenti a *Persona*.

### 1.5.2 Ereditarietà su entità: relationship

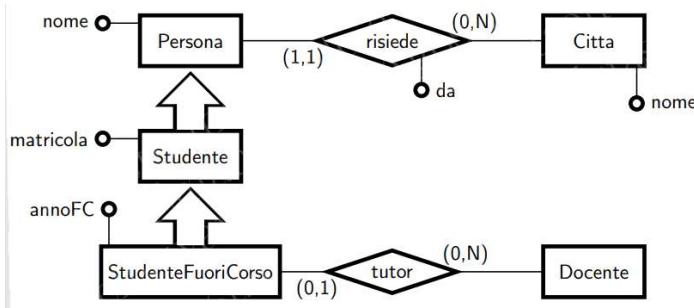


In questo schema possiamo osservare che ogni istanza di *Persona* deve essere coinvolta in esattamente una istanza di relationship *risiede*.

Siccome ogni istanza di *Studente* è anche (is-a) istanza di *Persona*, allora ogni istanza di *Studente* deve essere coinvolta in esattamente una istanza di relationship *risiede*.

Ovviamente l'entità *Studente* può essere coinvolta in ulteriori relationship.

### 1.5.3 Ereditarietà su entità: transitività

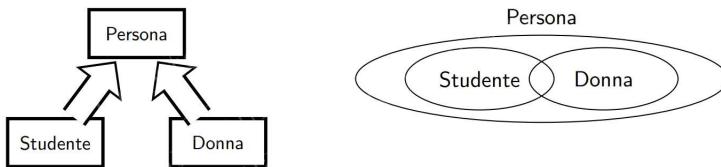


In questo schema notiamo che ogni istanza di *StudentFuoriCorso* è anche (*is-a*) istanza di *Studente*. Siccome ogni istanza di *Studente* è anche (*is-a*) istanza di *Persona*, allora ogni istanza di *StudentFuoriCorso*:

- Ha esattamente un valore per l'attributo *nome*
- Ha esattamente un valore per l'attributo *matricola*
- Deve essere coinvolta in esattamente una istanza di relationship *risiede* e può avere ulteriori proprietà (attributi o relationship).

### 1.5.4 Entità con più figlie

Un'entità può essere base di più relazioni *is-a*, le entità figlie possono avere *istanze in comune*, come ad esempio:

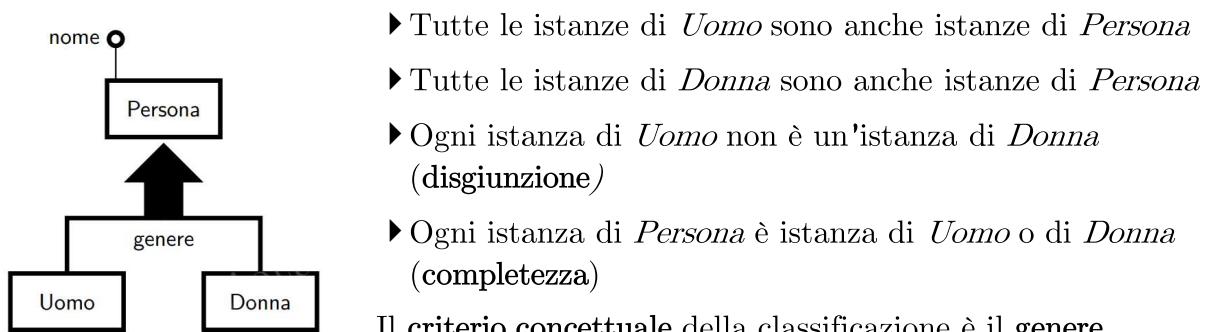


- Tutte le istanze di *Studente* sono anche istanze di *Persona*
- Tutte le istanze di *Donna* sono anche istanze di *Persona*
- Possono esistere istanze di *Persona* che non sono istanze né di *Studente* né di *Donna*
- Possono esistere istanze di *Persona* che sono istanze sia di *Studente* che di *Donna*

### 1.5.5 Generalizzazioni complete e non complete

ER offre un ulteriore costrutto rispetto alla relazione *is-a*: il costrutto della **generalizzazione**. La generalizzazione permette di classificare le istanze di una entità in più entità figlie secondo uno stesso criterio concettuale. Vediamo ora i due tipi di generalizzazione.

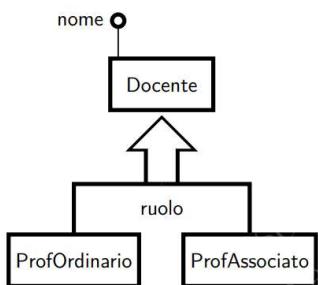
Generalizzazione completa:



Rappresentazione tramite diagramma di Eulero-Venn.

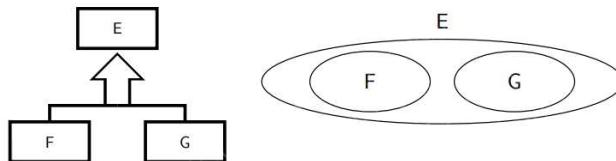


Generalizzazione non completa:



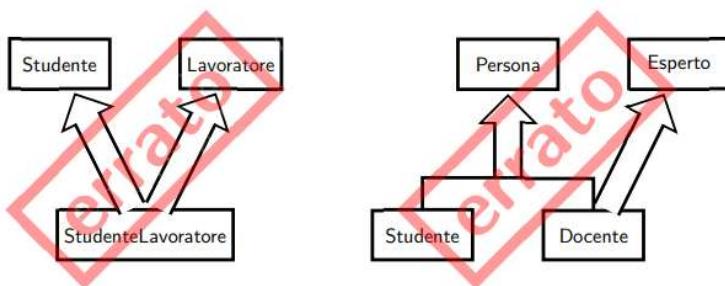
- ▶ Tutte le istanze di *ProfAssociato* sono anche istanze di *Docente*
- ▶ Tutte le istanze di *ProfOrdinario* sono anche istanze di *Docente I*
- ▶ Ogni istanza di *ProfAssociato* non è un'istanza di *ProfOrdinario* (*disgiunzione*)
- ▶ Possono esistere istanze di *Docente* che non sono istanze né di *ProfAssociato* né di *ProfOrdinario* (*nessun vincolo di completezza*)

Rappresentazione tramite diagramma di Eulero-Venn.

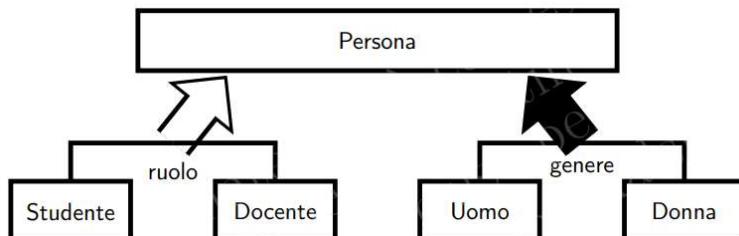


NB:

Un'entità non può essere coinvolta come figlia di più relazioni *is-a* e/o generalizzazioni. ER ammette solo ereditarietà singola.



### 1.5.6 Generalizzazioni multiple con stessa entità base

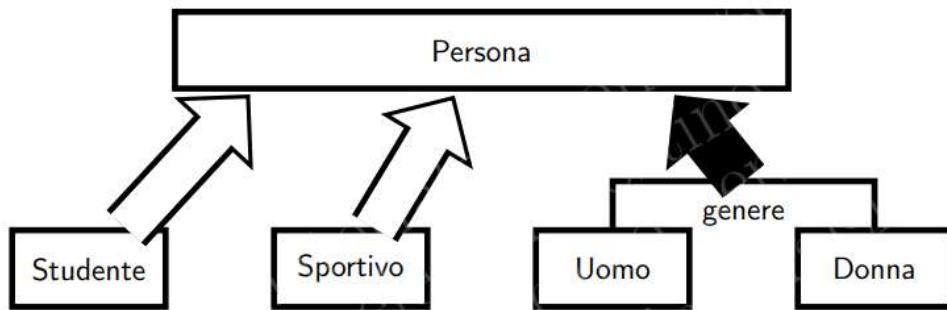


Analizziamo il diagramma proposto:

- Secondo il criterio del genere, le persone si partizionano in *uomini* e *donne* (*generalizzazione completa*)

- Non esistono persone che siano sia *uomini* che *donne* (*disgiunzione*) e non esistono persone che non sono *né uomini né donne* (*completezza*)
- Secondo il criterio del *ruolo*, le persone si classificano in *impiegati*, *studenti* e altri (*generalizzazione non completa*). Non esistono persone che siano *sia impiegati che studenti* (*disgiunzione*)
- Ogni istanza di *Impiegato* e ogni istanza di *Studente* sarà anche istanza di *esattamente una* tra *Uomo* e *Donna*.

### 1.5.7 Generalizzazioni e IS-A multiple con stessa entità base



Analizziamo il diagramma proposto:

- Secondo il criterio del genere, le persone si partizionano in *uomini* e *donne* (*generalizzazione completa*)
- Non esistono persone che siano sia *uomini* che *donne* (*disgiunzione*) e non esistono persone che non sono *né uomini né donne* (*completezza*)
- Alcune persone sono *studenti* (ed in quanto *persona*, ogni *studente* è o *uomo* o *donna*)
- Alcune persone sono *sportivi* (ed in quanto *persona*, ogni *sportivo* è o *uomo* o *donna*)

Possiamo chiaramente avere studenti sportivi uomini, studenti non sportivi donne, etc.

















