DIPARTIMENTO
DI INFORMATICA
SAPIENZA
UNIVERSITÀ DI ROMA

Basi di Dati II

Authors

Emanuele D'Agostino
Giuseppe Borracci

GitHub

Rurik-D
GiusTMP

AA 22/23

Indice

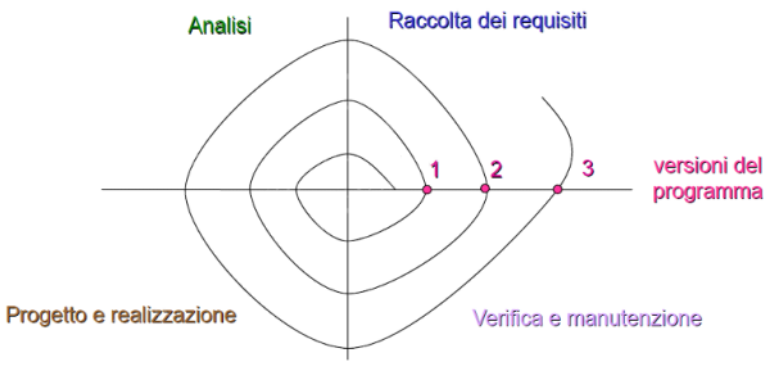
0 Introduzione

0.1 Ciclo di vita a cascata di un software

Un software può essere realizzando 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'obbiettivo 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)

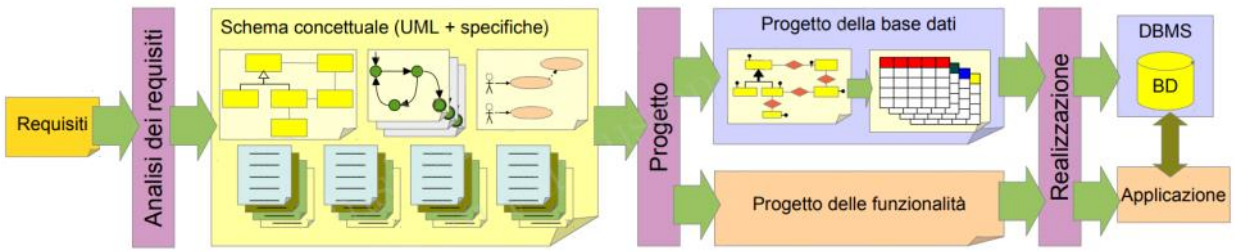
UML fornisce costrutti per modellare gran parte degli aspetti, sia statici che dinamici (dati, funzioni, evoluzione), usando un approccio Object Oriented.
Si tratta di un linguaggio estremamente vasto (ed anche estendibile), che permette di produrre diagrammi per descrivere:

- 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** (è l'astrazione di un concetto).

1.2.1 Attributo di entità

Ad una entità possono corrispondere degli **attributi**, che rappresentano delle **proprietà locali dell'entità** di interesse per il dominio applicativo, il quale associa ad ogni istanza di entità un valore in un certo dominio (il tipo dell'attributo).

Esempi:

- Attributi per l'entità *Impiegato* → nome (stringa), età, stipendio (intero positivo)
- 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).

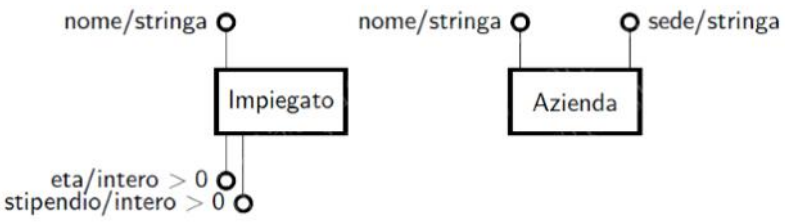
Ogni istanza di entità *Impiegato* ha (in questo esempio) associato:

- 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 istanza 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

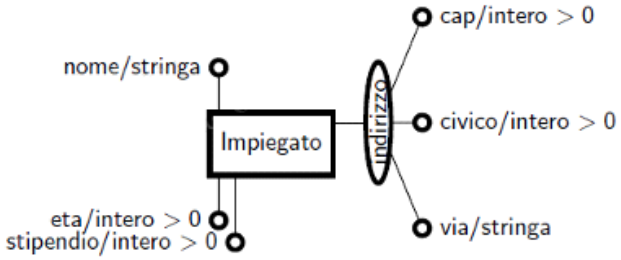
Per rendere la modellazione più aderente alla realtà, possiamo restringere i domini suddetti mediante **vincoli di dominio**:

- Definendo un limite, come ad esempio ***intero* > 0**, ***intero* ≥ 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 $\{uomo, donna\}$.

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**.

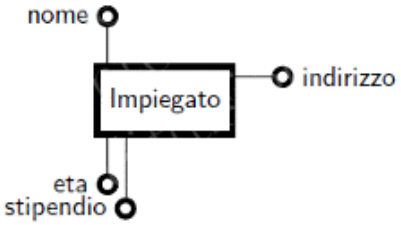


Per semplicità, i domini degli attributi non vengono indicati nel diagramma ER. Essi compaiono in un documento allegato.

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)

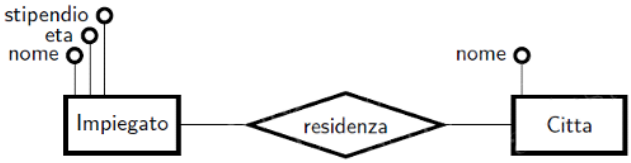
Anche i campi di un attributo composto possono essere omessi e indicati in un documento allegato.



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.

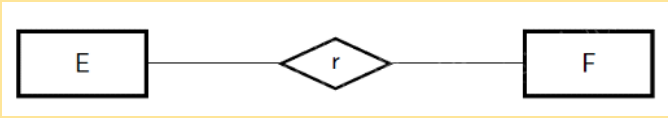
Poniamo ad esempio il caso di voler conoscere la città di residenza degli impiegati, esprimeremo la relationship come:



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:

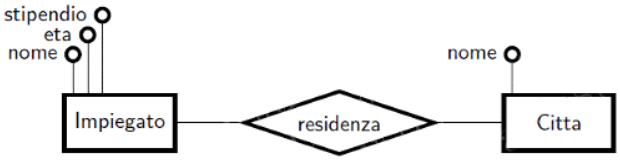
$$r \subseteq E \times F$$

- 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:

$$\text{Impiegato} = \{anna, mario\} \qquad \text{Citta} = \{milano, roma\}$$

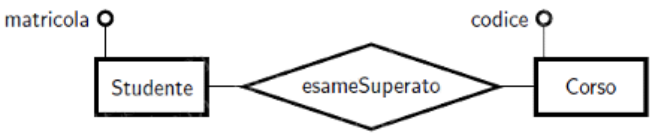
Prendiamo come esempio due possibili livelli estensionali per residenza:

$(anna, milano)$
$(mario, roma)$

$(anna, milano)$
$(mario, roma)$
$(anna, milano)$ ← duplicato, non ammesso!

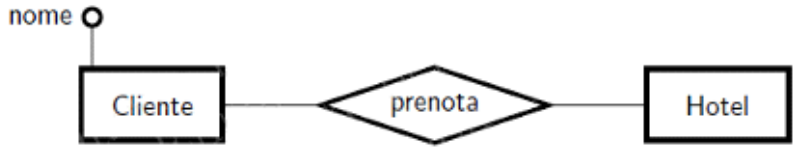
Non avrebbe infatti senso rappresentare due volte che l'impiegato di nome *Anna* risiede a *Milano*.

Esempio 2:



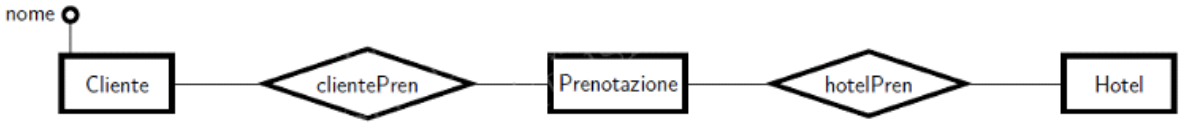
Grazie alla restrizione imposta dal **costrutto relationship**, stiamo affermando che *"uno stesso studente non può superare più volte lo stesso esame"*.

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**:



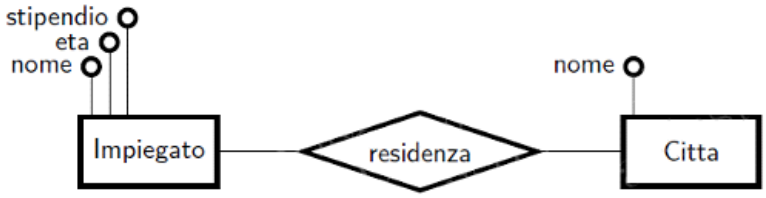
mario	*(mario, p1)*	*p1*	*(p1, hotel1)*	*hotel1*
anna	*(anna, p2)*	*p2*	*(p2, hotel2)*	*hotel2*
	(mario, p3)	*p3*	*(p3, hotel1)*	

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:

mario	*(mario, roma)*	*roma*
mario	*(mario, milano)*	*milano*

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 solo 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

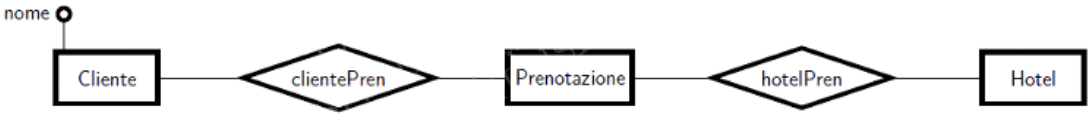
  Ogni impiegato ha una ed una sola città di residenza.

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

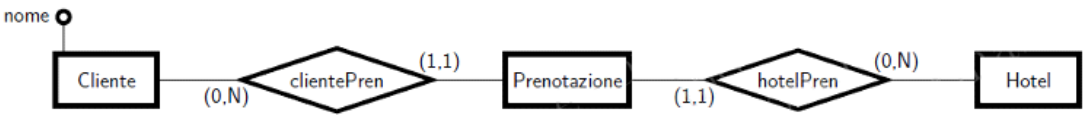
  Da 0 ad N ⇒ nessun limite

  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:

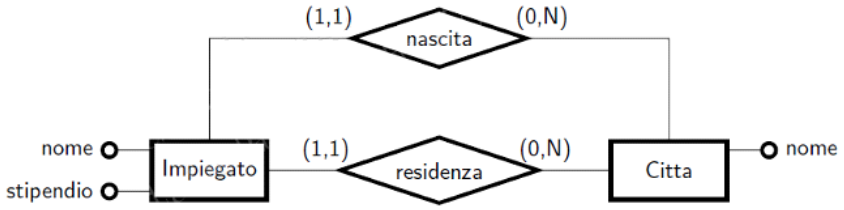


- Ogni istanza di Prenotazione deve essere coinvolta in:
  - esattamente una istanza di *clientePren* (perché è relativa ad *un* cliente)
  - esattamente una istanza di *hotelPren* (perché è relativa ad *un* hotel)
- Ogni istanza di Cliente può essere coinvolta in un numero arbitrario di istanze di *clientePren*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

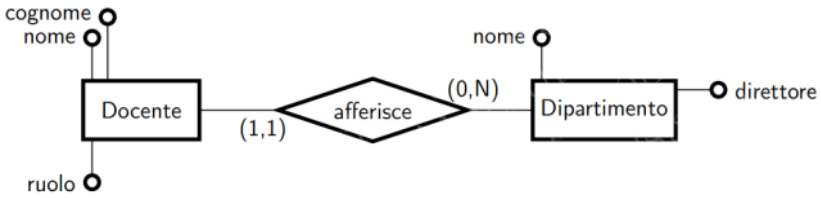
Analizziamo adesso un problema più pratico.

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**

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

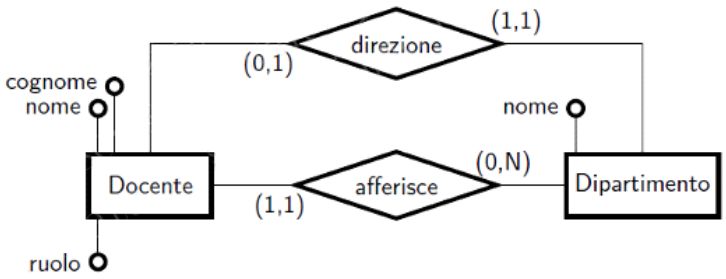
Entità **Dipartimento**

attributo	**dominio**
nome	stringa
direttore	stringa

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**

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

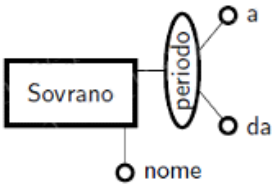
Entità **Dipartimento**

attributo	**dominio**
nome	stringa

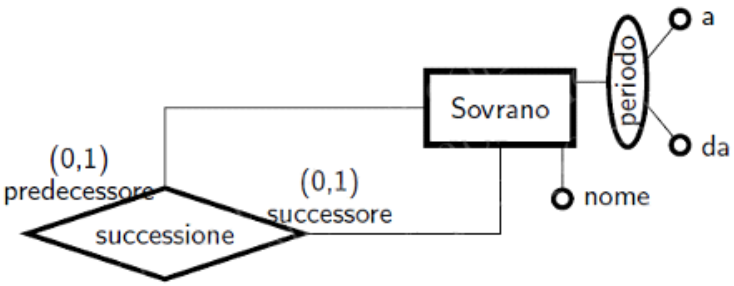
Notiamo infatti che *"un docente o non dirige niente o dirige al più un solo dipartimento* (0,1) *"* e che *"ogni dipartimento è gestito da uno e un solo direttore* (1, 1) *"*.

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 $(p1, p2) : p1, p2 \in Sovrano$.

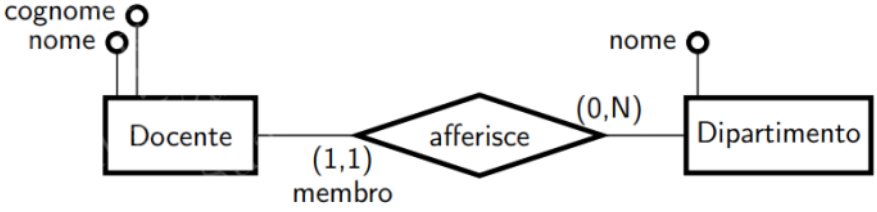
Per evitare ambiguità, diamo un nome ai ruoli che le istanze di entità Sovrano giocano nei legami (istanze di relationship).

Una istanza di relationship successione diventa una coppia etichettata:

$$(predecessore: p1,\ successore: p2)$$

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.

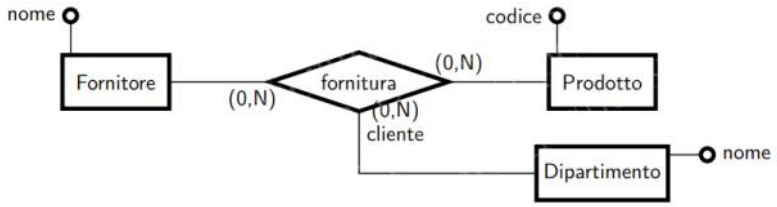


Se il ruolo di un'entità in una relationship non è indicato, si assume che abbia nome uguale a quello dell'entità. In questo caso il ruolo di Dipartimento in afferisce è: *Dipartimento.*

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 $E_1, E_2, \ldots E_n$ (non necessariamente tutte distinte), con ruoli, rispettivamente $u_1, u_2, \ldots u_n$ è costituita da un insieme di *n-ple* etichettate della forma

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

tali che

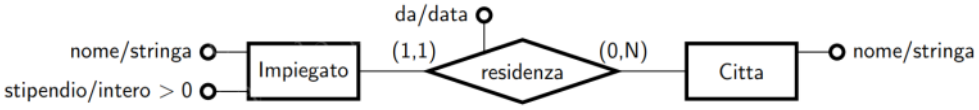
$$x_1 \in E_1, x_2 \in E_2, \ldots, x_n \in E_n \ .$$

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).

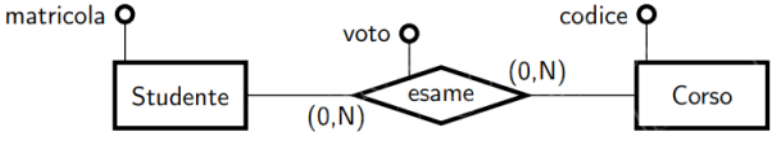
Il valore di un attributo di una istanza di relationship dipende solo dall'istanza stessa, non ha (in genere) alcun rapporto con le altre istanze .

Esempio 1:



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 2:



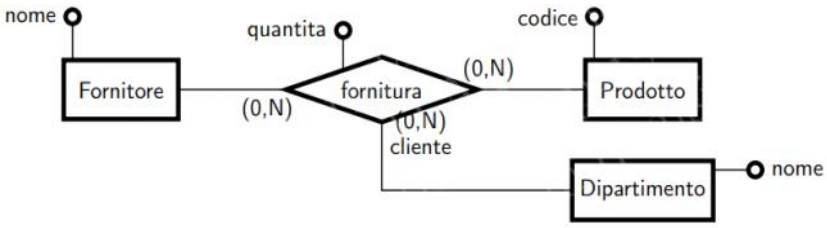
Entità *Studente*		Relationship *Esame*		Entità *Corso*	
Attributo	Dominio	Attributo	Dominio	Attributo	Dominio
matricola	intero	voto	[18, 30]	codice	intero

NB:

Un'istanza di relationship *esame* è ancora una coppia *(s, c)* con $s \in$ *Studente* e $c \in$ *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* <u>non</u> 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 1:



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

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

tali che

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

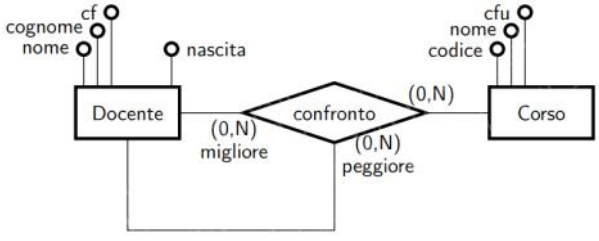
ad ognuna delle quali è associato un valore per l'attributo *quantità*.
Non possono inoltre coesistere due terne uguali (indipendentemente dal valore per l'attributo).

Esempio 2:

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 sia più apprezzato di quale altro come insegnante di un corso.



Entità *Docente*		Entità *Corso*	
Attributo	Dominio	Attributo	Dominio
nome	stringa	codice	intero
cognome	stringa	nome	stringa
cf	stringa	cfu	intero > 0
nascita	data		

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*.

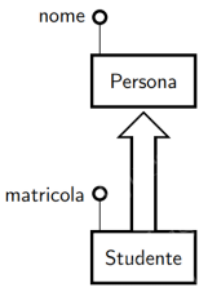
- *Persona* è l'**entità base**
- *Studente* è l'**entità derivata** (o **entità figlia**, o **sotto-entità**)

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 = $\{a, b, c, d\}$

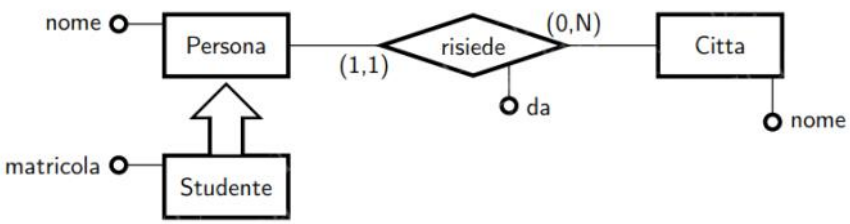
1.5.1 Ereditarietà su entità: attributi



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*.

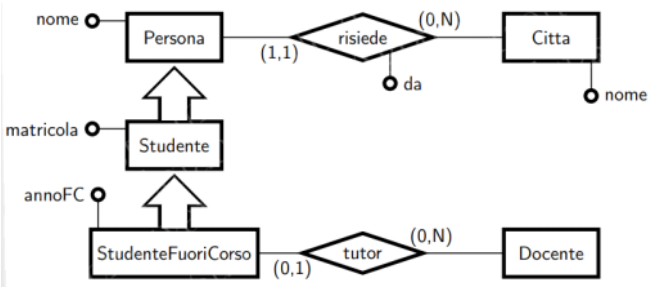
1.5.2 Ereditarietà su entità: relationship



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

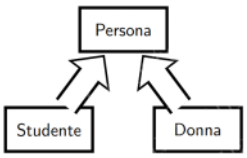


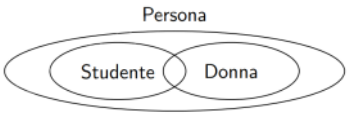
In questo schema notiamo che ogni istanza di *StudenteFuoriCorso* è anche (*is-a*) istanza di *Studente*. Siccome ogni istanza di *Studente* è anche (*is-a*) istanza di *Persona, allora* ogni istanza di S*tudenteFuoriCorso*:

- 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:



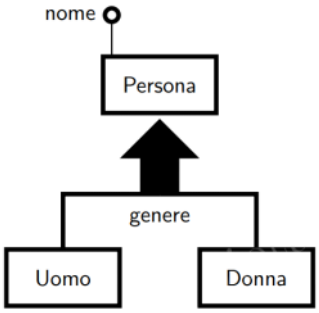


- Tutte le istanze di *Studente* sono anche istanze di *Persona*
- Tutte le istanze di *Donna* sono anche istanze di *Persona*
- Possono esistere istanze di *Persona* che non sono istanze né di *Studente* né di *Donna*
- Possono esistere istanze di *Persona* che sono istanze sia di *Studente* che di *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**. Vedremo ora i due tipi di generalizzazione.

Generalizzazione **completa**:

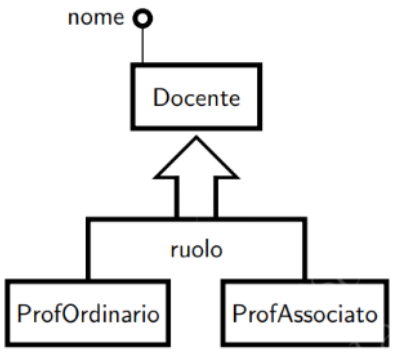


- Tutte le istanze di *Uomo* sono anche istanze di *Persona*
- Tutte le istanze di *Donna* sono anche istanze di *Persona*
- Ogni istanza di *Uomo* non è un'istanza di *Donna* (**disgiunzione**)
- Ogni istanza di *Persona* è istanza di *Uomo* o di *Donna* (**completezza**)

Il **criterio concettuale** della classificazione è il **genere**.

Rappresentazione tramite diagramma di Eulero-Venn:

Generalizzazione **non completa**:

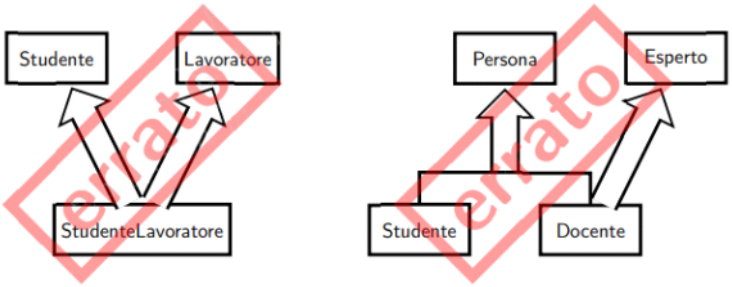


- Tutte le istanze di *ProfAssociato* sono anche istanze di *Docente*
- Tutte le istanze di *ProfOrdinario* sono anche istanze di *Docente* I
- Ogni istanza di *ProfAssociato* non è un'istanza di *ProfOrdinario* (**disgiunzione**)
- Possono esistere istanze di *Docente* che non sono istanze né di *ProfAssociato*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

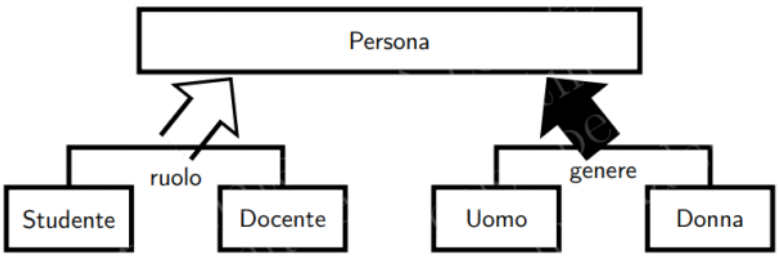
Rappresentazione tramite diagramma di Eulero-Venn.

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

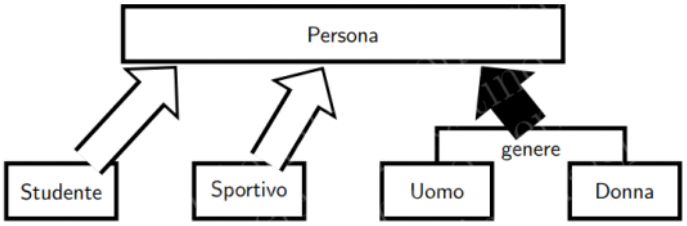


Analizziamo il diagramma proposto:

- 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*)

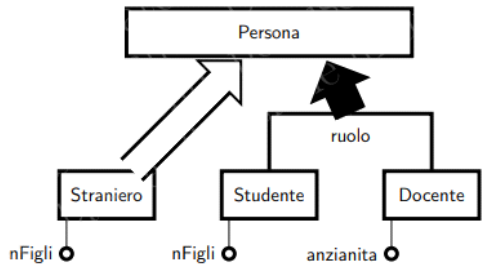
Possiamo chiaramente avere studenti sportivi uomini, studenti non sportivi donne, etc.

1.5.8 Attributi omonimi in generalizzazioni e is-a

In generale, **attributi con lo stesso nome associati ad entità diverse sono scorrelati**, cioè rappresentano funzioni diverse dalle istanze dell'entità ai valori del loro domino.
Ad esempio gli attributi nome di *Impiegato* e *Citta* (nel diagramma seguente) sono scorrelati:

Questo perché *Impiegato* e *Citta* non hanno istanze in comune.

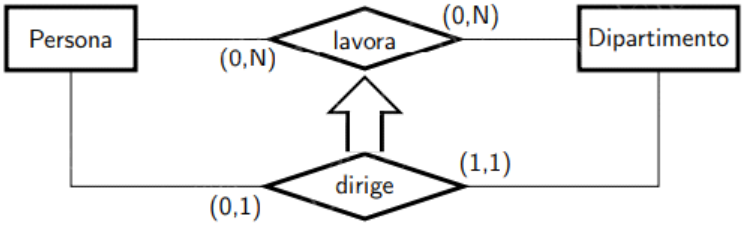
Quando due entità con attributi omonimi possono avere istanze in comune, si assume che tali attributi rappresentino la stessa funzione:



Uno studente straniero ha un unico valore per l'attributo ***nFigli***. Attributi omonimi di entità che possono avere istanze in comune devono anche avere lo stesso dominio.

1.5 Relazioni IS-A tra relationship

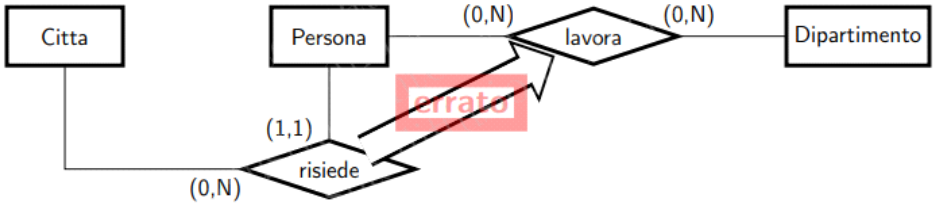
ER permette di modellare relazioni is-a tra relationship. Esempio:



Le persone possono lavorare in dipartimenti, ogni dipartimento ha un direttore. Ogni persona può dirigere al più un dipartimento e ogni direttore deve lavorare nel dipartimento che dirige (oltre, eventualmente, in altri).

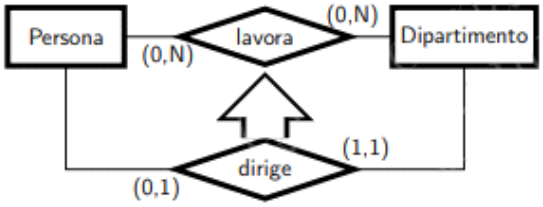
Semantica: ogni istanza (Persona:p, Dipartimento:d) di *dirige* è anche istanza di *lavora*

Bisogna però stare molto attenti, perché è facile modellare cose senza senso:



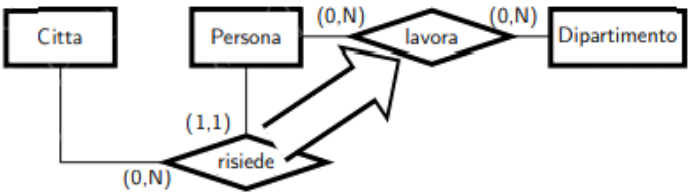
Le istanze di relationship risiede sono del tipo (Citta:c, Persona:p), mentre le istanze di relationship lavora sono del tipo (Persona:p, Dipartimento:d): la relazione is-a non può valere!

Quando ha senso che una relazione is-a tra relationship sia definita?



Sì

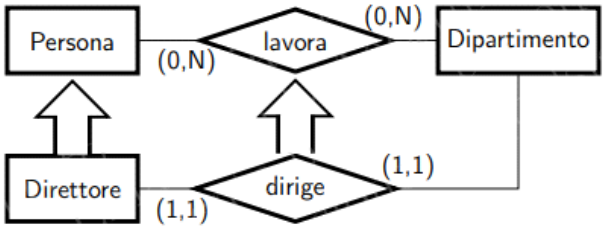
dirige e *lavora* sono dello stesso tipo!



No

risiede e *lavora* non sono dello stesso tipo!

E in questo caso?



Sebbene le relationship lavora e dirige non siano esattamente dello stesso tipo, ha ancora senso parlare di relazione is-a tra loro. Questo perché Direttore is-a Persona, quindi il tipo di lavora (Persona × Dipartimento) è una estensione del tipo di dirige (Direttore × Dipartimento):

$$(\text{Direttore} \times \text{Dipartimento}) \subseteq (\text{Persona} \times \text{Dipartimento})$$

1.5.8 Relazioni is-a tra relationship ben definite

Vedremo ora le condizioni che devono essere rispettate affinché la semantica di r *is-a* q sia ben definita.

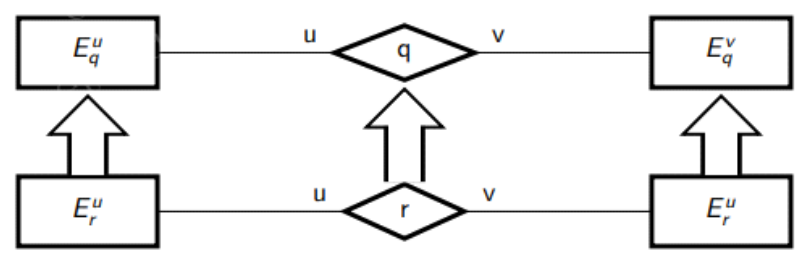
Condizione 1: *tipo di* $r \subseteq$ *tipo di* q

- r e q hanno la stessa arità
- r e q hanno gli stessi ruoli
- $\forall$ ruolo u, siano E_r^u e E_q^u le entità corrispondenti ad u in r e q, si ha

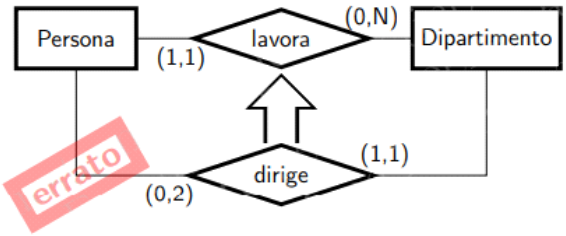
$$E_r^u = E_q^u$$

oppure

$$E_r^u \textit{ discende da } E_q^u$$

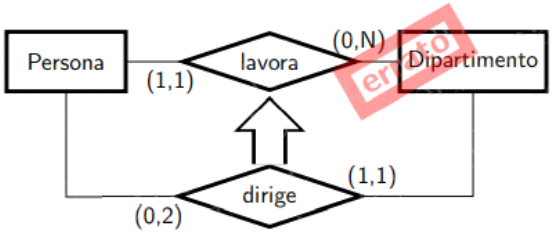


Esempio 1:



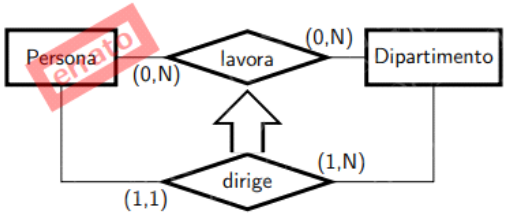
1. Ogni *impiegato* lavora in esattamente un *dipartimento*
2. Ogni *impiegato* può essere direttore di al più due *dipartimenti*
3. I *direttori* devono lavorare nei dipartimenti che dirigono: il valore massimo del vincolo di cardinalità *(0,2)* non può essere mai raggiunto e può essere ridotto ad 1 producendo un diagramma equivalente ma più accurato

Esempio 2:



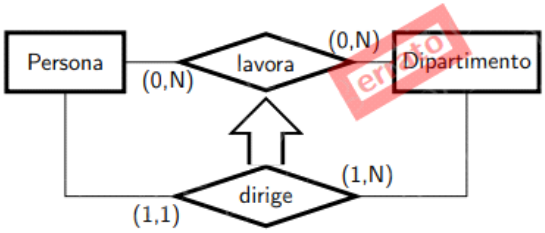
1. In ogni *dipartimento* può lavorare un qualunque numero *(0,N)* di *persone*.
2. Ogni *dipartimento* ha esattamente un *direttore*.
3. I *direttori* devono lavorare nei dipartimenti che dirigono: il valore minimo del vincolo di cardinalità *(0,N)* non può essere mai raggiunto e può essere aumentato ad 1 producendo un diagramma equivalente ma più accurato.

Esempio 3:



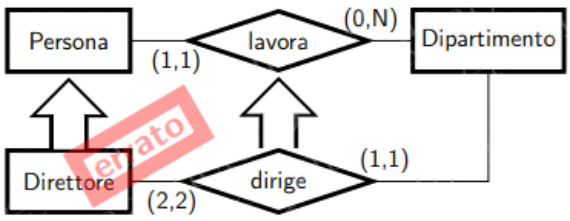
1. Ogni *impiegato* può lavorare in un numero arbitrario di *dipartimenti* (anche nessuno).
2. Ogni *impiegato* è sicuramente direttore di un *dipartimento*.
3. I *direttori* devono lavorare nei dipartimenti che dirigono: il valore minimo del vincolo di cardinalità *(0,N)* non può essere mai raggiunto e può essere aumentato ad 1 producendo un diagramma equivalente ma più accurato

Esempio 4:



1. Nei *dipartimenti* possono lavorare un numero arbitrario di impiegati (anche nessuno).
2. Ogni *dipartimento* ha almeno un *direttore*.
3. I direttori devono lavorare nei dipartimenti che dirigono: il valore minimo del vincolo di cardinalità *(0,N)* non può essere mai raggiunto e può essere aumentato ad 1 producendo un diagramma equivalente ma più accurato.

Esempio 5:



1. Ogni *impiegato* lavora in esattamente un *dipartimento*
2. Ogni *direttore* dirige esattamente due *dipartimenti*
3. I direttori devono lavorare nei dipartimenti che dirigono: le uniche istanze ammissibili non definiscono alcun direttore, e quindi nessun dipartimento, e quindi nessuna persona.
   L'unica istanza ammissibile per il diagramma è quella vuota!

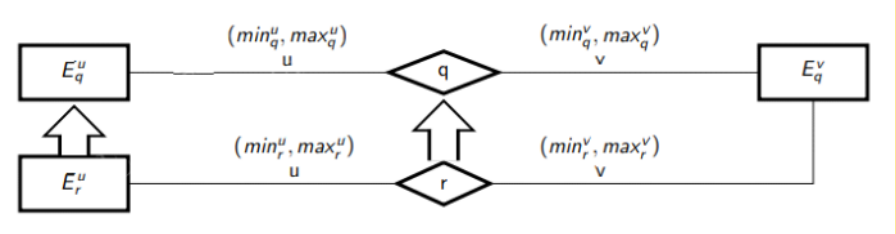
Condizione 2: *r is-a q tra relationship*

Per ogni ruolo u (che per la Condizione 1 è comune a r e q), siano E_r^u e E_q^u le entità corrispondenti ad u in r e q (per Condizione 1 si ha:

$$E_r^u = E_q^u \text{ oppure } E_r^u \text{ is-a } E_q^u \text{).}$$

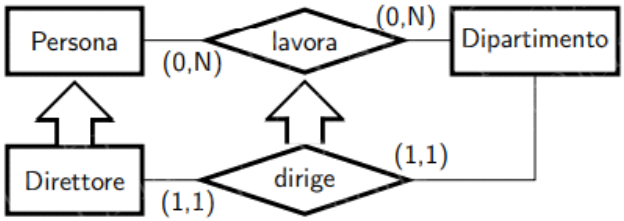
Siano $(\min_r^u, \max_r^u)$ e $(\min_q^u, \max_q^u)$ i vincoli di cardinalità per il ruolo u in r e q, si deve avere:

- $\max_r^u \leq \max_q^u$
- se $E_r^u = E_q^u$, allora $\min_r^u \leq \min_q^u$



Nota: una relazione is-a tra relationship che viola Condizione 2 è formalmente ben definita, ma evidenzia un errore di modellazione.

Esempio:



In questo diagramma, la relazione is-a tra le relationship dirige e lavora soddisfa la condizione 1, ma viola la condizione 2 (card. minime):

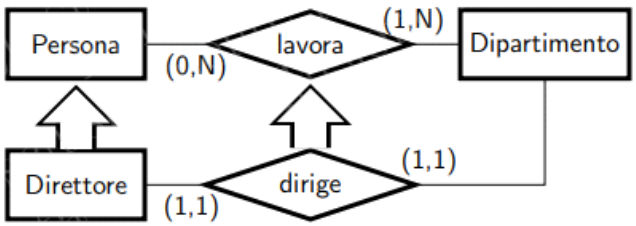
- Il vincolo di cardinalità minima per il ruolo Dipartimento in dirige è 1
- Il vincolo di cardinalità minima per il ruolo Dipartimento in lavora è 0

Stiamo affermando che:

1. Ogni *direttore* di dipartimento deve lavorare in quel *dipartimento* (relazione is-a)
2. Ogni dipartimento ha esattamente un *direttore*
3. Ogni dipartimento può avere un numero arbitrario (anche zero) di persone che vi lavorano

La relazione is-a tra relationship è ben definita, ma evidenzia una inaccuratezza nella modellazione: ogni dipartimento deve avere almeno una persona che vi lavora (il suo *direttore*)!

È possibile ottenere un diagramma equivalente ma più accurato aumentando il vincolo di cardinalità minima per il ruolo *Dipartimento* in *lavora* ad 1 (la condizione 2 sarebbe soddisfatta).
Quindi, dopo il raffinamento del vincolo di cardinalità di *Dipartimento* in *lavora* abbiamo:



E per i ruoli di *Persona/Direttore* in *lavora/dirige*?

Sebbene il vincolo di cardinalità minima di *Direttore* in *dirige* (1) sia maggiore di quello di *Persona* in *lavora* (0), la condizione 2 è soddisfatta perché *Direttore* e *Persona* non sono la stessa entità:

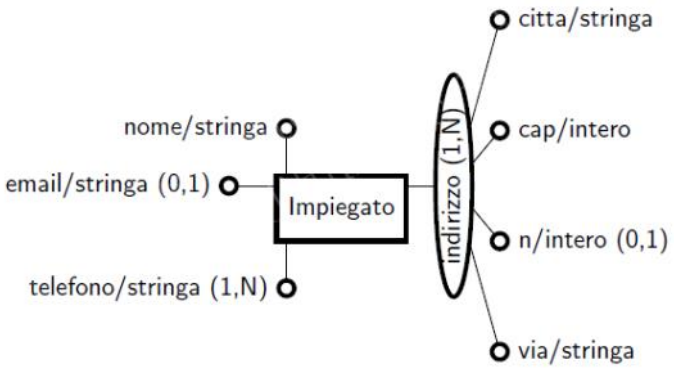
- *Ogni direttore* dirige esattamente *un dipartimento*
- Possono esistere persone (non direttori) che non lavorano in alcun dipartimento

Nota: la condizione 2 vincola la molteplicità minima di un ruolo u di r (r *is-a* q) ad essere al più quella di q solo se r e q coinvolgono la stessa entità nel ruolo u.

1.6 Vincoli di cardinalità sugli attributi

ER permette di definire vincoli di **cardinalità sugli attributi** di entità e relationship. Possiamo quindi rappresentare:

- Opzionalità
- Valori multipli



Se il vincolo di cardinalità di un attributo non è indicato, è da intendersi (1,1) (valore singolo e obbligatorio).

1.7 Vincoli di Identificazione di Entità

ER permette di esprimere vincoli di integrità sul diagramma, che restringono l'insieme dei livelli estensionali ammessi. Abbiamo già visto alcuni vincoli di integrità: i vincoli di cardinalità per attributi e relationship.

Presentiamo ora i **vincoli di identificazione di entità**.

Un identificatore per una entità E è un insieme I di attributi e/o ruoli di relationship (tutti a cardinalità (1,1)) in cui E è coinvolta tale che:

- **Non esistono due istanze di *E* che coincidono in tutti i valori per *I***
- **Un vincolo di identificazione sull'entità *E* definisce un identificatore**

1.7.1 Vincoli di identificazione di entità interni

Un **vincolo di identificazione interno** per un'entità è definito su un insieme di soli attributi dell'entità.

Esempio:

1. Si vogliono rappresentare studenti, con nome, cognome, matricola e codice
2. Non esistono due studenti con lo stesso codice fiscale
3. Non esistono due studenti con la stessa matricola

I requisiti 2. e 3. richiedono di definire gli identificatori interni *{cf}* e *{matricola}* per l'entità *Studente*.

Un **vincolo di identificazione interno** su un identificatore consistente in un singolo attributo. Si definisce annerendo il cerchio relativo a quell'attributo.

Esempio:

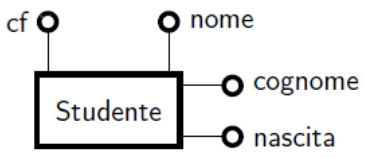
1. Si vogliono rappresentare studenti, con nome, cognome, matricola e codice fiscale
2. Non esistono due studenti con lo stesso codice fiscale
3. Non esistono due studenti con la stessa matricola

Un vincolo di identificazione interno **può coinvolgere più attributi**.

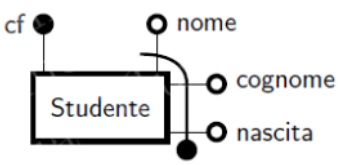
Esempio:

- Si vogliono rappresentare studenti, con nome, cognome, matricola, codice fiscale e data di nascita.
- Non esistono due studenti con lo stesso codice fiscale.
- Non esistono due studenti che coincidono nel nome, cognome e data di nascita (è solo un esempio, non è un vincolo valido in generale).

I requisiti 2. e 3. richiedono di definire gli identificatori interni *{cf}* e *{nome; cognome; nascita}* per l'entità Studente



Un **vincolo di identificazione su un identificatore** (interno) consistente in attributi multipli si definisce come in figura:



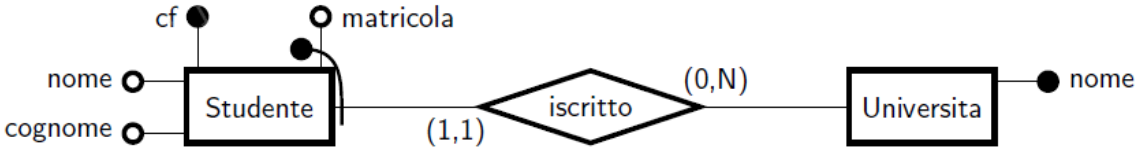
Nota: un attributo può essere coinvolto in identificatori multipli.

1.6.1 Vincoli di identificazione di entità esterni

Un **vincolo di identificazione esterno** coinvolge attributi e ruoli di relationship, tutti a molteplicità (1,1).

Esempio:

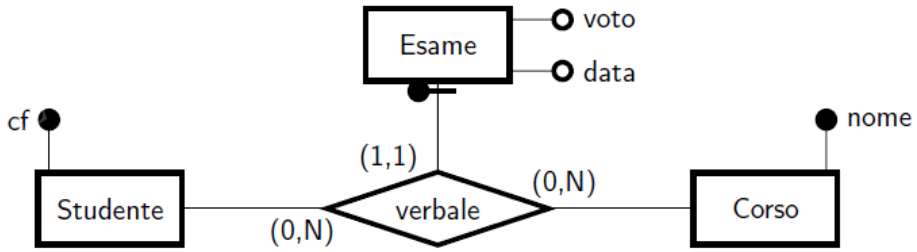
- Si vogliono rappresentare studenti (con matricola e codice fiscale) e università (con nome)
- Non esistono due studenti con lo stesso codice fiscale
- Non esistono due studenti della stessa università con la stessa matricola
- Non esistono due università con lo stesso nome



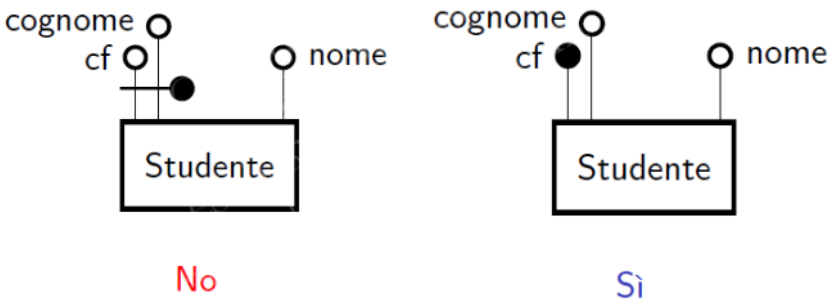
Un vincolo di identificazione esterno può coinvolgere anche un solo ruolo di relationship.

Esempio:

- Si vogliono rappresentare studenti, corsi ed esami
- Non esistono due studenti con lo stesso codice fiscale
- Non esistono due corsi con lo stesso nome
- Non esistono due esami per la stessa coppia studente/corso



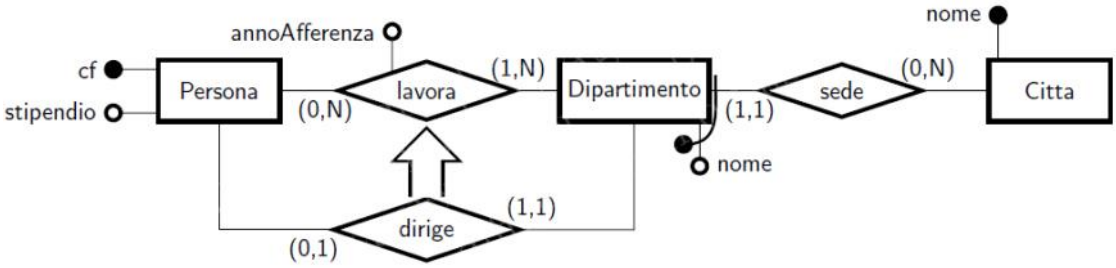
I vincoli di identificazione devono essere **⊆-minimali**:



1.7 Vincoli Esterni al diagramma ER (Informale)

Spesso è necessario imporre ulteriori vincoli di integrità, che non sono esprimibili direttamente in ER (business rules). Vedremo adesso un metodo informale per esprimerli (più avanti vedremo una metodologia più rigorosa).

Esempio:



1. Ogni direttore deve lavorare da ≥ 5 anni nel dipartimento che dirige
2. Nessun impiegato può avere uno stipendio superiore a quello del direttore del suo dipartimento
3. Il direttore in ogni dipartimento con sede a Roma deve avere almeno 10 anni di anzianità in quel dipartimento.

V.dirige.afferenza:

Per ogni istanza *(p: Persona; d: Dipartimento)* della relationship *dirige*, l'istanza *(p: Persona; d: Dipartimento)* della relationship *lavora* deve avere un valore v per l'attributo *annoAfferenza* per cui vale:

$$v \leq \ annoCorrente \ - 5$$

dove *annoCorrente* denota l'istanza del tipo intero che rappresenta l'anno corrente.

V.Persona.stipendio:

Per ogni istanza *(dir : Persona; dip : Dipartimento)* della relationship *dirige* e per ogni istanza *(p : Persona; dip : Dipartimento)* della relationship *lavora* relativa ad uno stesso dipartimento dip, siano:

$stip_{dir}$ il valore dell'attributo stipendio di *dir*

$stip_p$ il valore dell'attributo stipendio di p

Deve essere:

$$stipdir \geq stipp$$

V.dirige.Roma:

Per ogni coppia di istanze *(dir : Persona; dip : Dipartimento)* della relationship dirige e *(dip : Dipartimento; c : Citta)* della relationship sede relative ad uno stesso dipartimento dip, se l'istanza *c : Citta* ha come valore dell'attributo *nome* la stringa "Roma", allora il valore a dell'attributo afferenza dell'istanza *(dir : Persona; dip : Dipartimento)* della relationship *lavora* deve essere tale che:

$$a \leq \ annoCorrente \ - 10$$

dove *annoCorrente* denota l'istanza del tipo intero che rappresenta l'anno corrente.

2 Diagrammi UML degli Use-Case

2.1 Concetti Base

Lo **use-case diagram**, in italiano *diagramma dei casi d'uso*, fa parte dei diagrammi comportamentali dell'Unified Modelling Language.
Con uno use-case diagram vengono rappresentate le **funzioni di un sistema dal punto di vista dell'utilizzatore**, chiamato in UML **attore**. Questo attore non deve per forza essere un utente umano; tale ruolo può anche essere attribuito ad un sistema esterno che ha accesso ad un altro sistema.

> Lo use-case diagram rappresenta la **relazione tra un attore e le sue richieste o aspettative rispetto al sistema**, senza descrivere le azioni in corso di svolgimento o metterle in una sequenza logica.

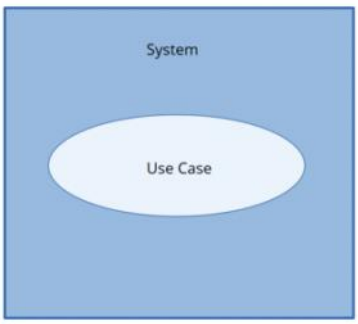
Questa struttura nella pratica è adatta a rappresentare le funzioni principali e/o gli obiettivi di un sistema in maniera chiara. Per questo motivo, nello **sviluppo dei software o nella pianificazione di nuovi processi aziendali,** spesso uno dei primi passi è quello di creare uno use-case diagram, per definire insieme al committente cosa dovrà fare il sistema.

2.1.1 Componenti

Al fine di rendere chiaro a colpo d'occhio lo use case diagram, vengono utilizzate componenti standardizzate per la rappresentazione. Esse modellano le funzionalità che il sistema deve realizzare, in termini di use-case (scenari di utilizzo). Ci sono due elementi essenziali: **attore** e **use-case**.

> **Definizione: Attore**
>
> Ruolo che un utente (umano o sistema esterno) gioca interagendo con il sistema. Lo stesso utente può essere rappresentato da più attori (può giocare più ruoli). Più utenti possono essere rappresentati dallo stesso attore.

> **Definizione: Use-ase**
>
> Cattura un insieme omogeneo di funzionalità accedute da un gruppo omogeneo di utenti. Tipicamente coinvolge concetti rappresentati da più entità e relationship del diagramma ER.

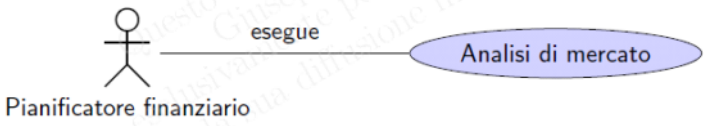


Lo **use-case diagram** è quindi un grafo in cui i nodi rappresentano attori e use-case e gli archi rappresentano:

- La possibilità per un attore di invocare uno use-case
- La possibilità per uno use-case di invocare un altro use-case
- La generalizzazione tra attori e tra use-case

2.1.2 Associazione

Un'associazione è un'altra componente dello use-case diagram. Essa modella la possibilità di accesso, da parte di un attore, alle funzionalità di uno use-case.



Il nome dell'associazione è opzionale.

2.2 Inclusione ed Estensione

Per la relazione tra diversi casi d'uso si utilizza un diverso tipo di associazione, rappresentata da righe tratteggiate. Esistono **diversi tipi di associazioni tra use-case** difatti, se necessario, le linee possono avvalersi di frecce e **stereotipi** (parole chiave rappresentate tra doppie parentesi angolate) per chiarire le relazioni tra i singoli elementi.
Distinguiamo due stereotipi: **include** e **extend**.

Inclusione

Modella il concetto che ogni istanza di Use-case A includerà il comportamento di Use-case B.

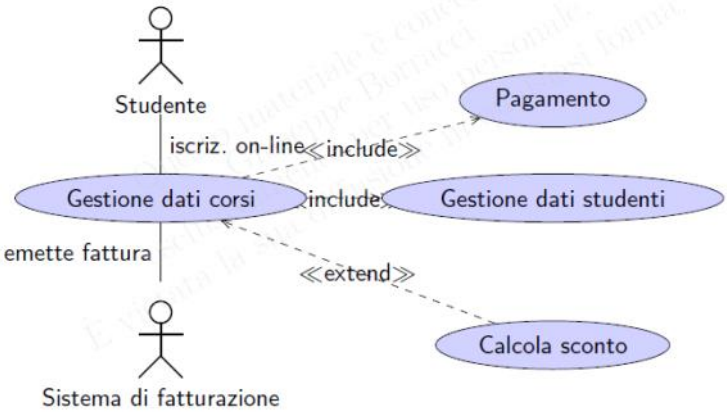
In altre parole: qualche funzionalità di Use-case A ha bisogno di servirsi di qualche funzionalità di Use-case B.

Estensione

Modella il concetto che le istanze di Use-case C, in casi particolari, sono estese aggiungendo il comportamento di Use-case D.

In altre parole: qualche funzionalità di Use-case C in casi particolari ha bisogno di essere estesa con qualche funzionalità di Use-case D.

Vediamo un esempio di applicazione dei concetti visti fin ora:



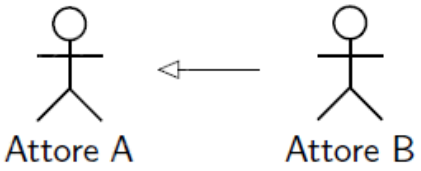
2.3 Generalizzazione

Vedremo ora un caso particolare di relazione, applicabile sia tra due attore che tra due use-case: la **generalizzazione**.

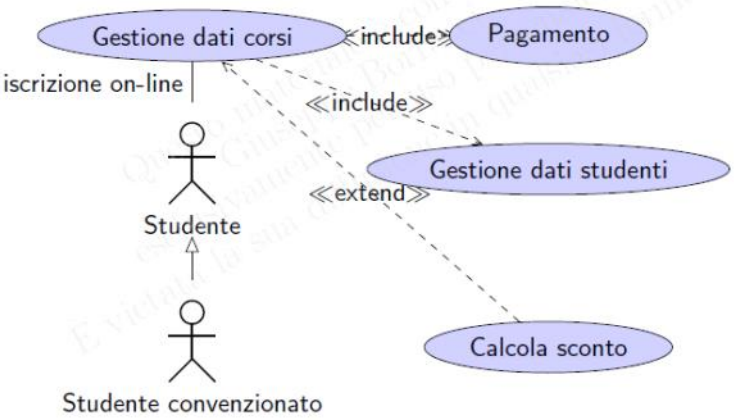
Tale relazione rappresenta il concetto per il quale uno use-case (o un attore) sia un caso particolare di un altro use-case (o di un altro attore).

2.3.1 Generalizzazione tra attori

Dati due attori A e B, diremo che l'attore B (**sottoattore**) è un caso particolare dell'attore A (**superattore**).

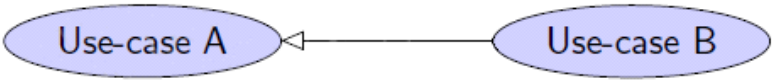


Esempio:

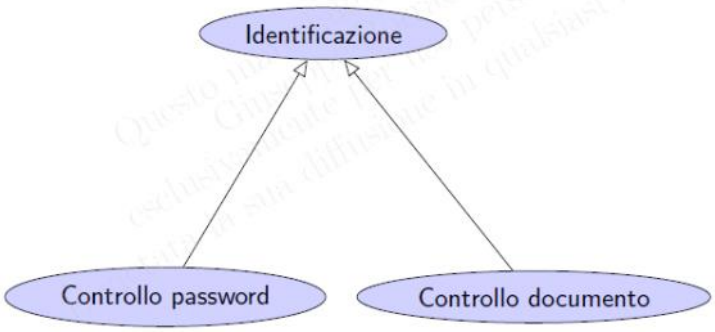


2.3.2 Generalizzazione use-case

Dati due use-case A e B, diremo che lo use-case B è un caso particolare dello use-case A.



Esempio:



2.4 Specifiche di use-case informali

Il diagramma UML degli use-case non definisce esattamente le funzionalità che rappresenta, ciò comporta che ogni use-case diagram viene corredato da un **documento di specifica**.

La specifica di ogni singola operazione di use-case è strutturata nel modo seguente:

> ***operazione*$_z$*(arg*$_1$*: dom*$_1$*, ... , arg*$_n$*: dom*$_n$*): dom*$_{rit}$
>
> **precondizioni**: pre-condizioni
>
> **postcondizioni**: post-condizioni

segnatura: Nome dell'operazione, nome e dominio degli eventuali argomenti e dominio dell'eventuale valore di ritorno.

precondizioni: Condizioni sugli **argomenti** e sul **livello estensionale** del sistema che devono valere all'**avvio** dell'operazione, affinché il suo **comportamento sia ben definito**.

postcondizioni: Condizioni sul **livello estensionale** del sistema che devono valere al **termine** dell'esecuzione dell'operazione (nel caso questa faccia **side-effect**) e definizione dell'eventuale valore di ritorno.

Struttura della specifica use-case:

nome_use-case

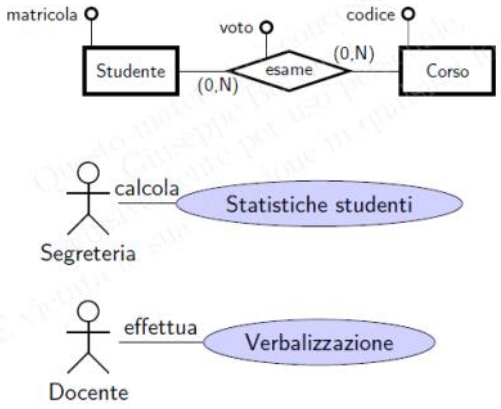
> ***operazione*$_1$*(arg*$_1$*: dom*$_1$*, ... , arg*$_n$*: dom*$_n$*): dom*$_{rit}$
>
> **precondizioni**: pre-condizioni
>
> **postcondizioni**: post-condizioni

⋮

> ***operazione*$_k$*(arg*$_1$*: dom*$_1$*, ... , arg*$_n$*: dom*$_n$*): dom*$_{rit}$
>
> **precondizioni**: pre-condizioni
>
> **postcondizioni**: post-condizioni

END

Esempio:



Specifica use-case *Statistiche studenti*:

> ***mediaVoti(s : Studente) : reale in [18,31]***
>
> **precondizioni**: L'istanza s è coinvolta in almeno un'istanza della relationship esame.
>
> **postcondizioni**: result è la somma dei valori dell'attributo voto di tutte le istanze di relationship esame definite nel livello estensionale nelle quali l'istanza s è coinvolta, diviso per il numero di tali istanze.

> ***numMedioEsami() : reale ≥ 0***
>
> **precondizioni**: Il livello estensionale dei dati definisce almeno una istanza di entità Studente.
>
> **postcondizioni**: result è pari al numero di istanze di relationship esame definite nel livello estensionale diviso per il numero di istanze di entità Studente.

END

Specifica use-case *Verbalizzazione*:

> ***verbalizzaEsame(s: Studente, c: Corso, v: [18,31])***
>
> **precondizioni**: L'istanza s non è coinvolta in alcuna istanza della relationship esame con l'istanza c.
>
> **postcondizioni**: Viene creata l'istanza (s : Studente; c : Corso) della relationship esame con valore v per l'attributo voto.

END

Nota: nella specifica denotiamo con result il risultato dell'operazione.

2.4.2 Specifiche di use-case e linguaggio naturale

Come per la definizione dei vincoli esterni al diagramma ER, anche per la specifica delle operazioni di use-case l'uso del linguaggio naturale è pericoloso, in quanto: potenzialmente ambiguo, potenzialmente omissivo, potenzialmente contraddittorio, in generale poco leggibile (soprattutto per operazioni di una certa complessità).

Vedremo successivamente come definire la specifica delle operazioni use-case in modo formale utilizzando la logica matematica.