SAPIENZA
UNIVERSITÀ DI ROMA

UNIVERSITÀ "SAPIENZA" DI ROMA
FACOLTÀ DI INFORMATICA

Basi di Dati II

Appunti integrati con il libro "Software Engineering, a Practitioner's Approach", R. Pressman, B.R. Maxim

Author
Simone Bianco

7 marzo 2023

Indice

Capitolo 0

Introduzione

Capitolo 1

Cenni di Ingegneria del Software

1.1 Obiettivi e Contesto organizzativo

Durante lo sviluppo di software complessi, è impossibile passare direttamente allo sviluppo dell'applicazione senza prima effettuare un'attenta **analisi delle necessità** e delle **modalità di realizzazione** del software stesso.

Consideriamo i seguenti due esempi:

- Vogliamo progettare un database in grado di memorizzare informazioni su un insieme di contatti (telefonici e email).

  Per ogni contatto è necessario mantenere:

  - Nome e cognome
  - Numeri di telefono di casa, ufficio e mobile
  - Indirizzo email

  I contatti possono appartenere a gruppi. L'applicazione deve permettere all'utente di aggiungere un nuovo contatto, modificare i dati di un contatto, cancellare un contatto, assegnare/rimuovere contatti a/da un gruppo e ricercare i contatti per nome e/o cognome

- Si vuole sviluppare un sistema che permetta ad una banca di gestire i conti correnti dei clienti, i loro investimenti, oltre che la propria rete di promotori finanziari.

  Il sistema deve tenere traccia di tutti gli acquisti e vendite di azioni, obbligazioni, etc. effettuati dai clienti, e deve poter calcolare in tempo reale la valorizzazione corrente del loro portafoglio.

  Inoltre, l'applicazione deve assistere i promotori finanziari nella scelta degli strumenti finanziari più adeguati da proporre ai clienti, e deve permettere ai responsabili di agenzia di controllare la professionalità dei promotori.

In questo caso, il primo esempio, trattandosi di un programma semplice e avente poche funzionalità, non richiede alcuna progettazione. Tuttavia, realizzare il secondo esempio senza effettuare una rigorosa analisi del software risulta essere **impossibile**.

Statisticamente, difatti, il progetto di un software complesso richiede:

- Una pool di ingegneri del software, progettisti, analisti e programmatori
- Circa **6 mesi** per l'analisi delle richieste
- Circa **9 mesi** per la progettazione del software
- Circa **3 mesi** per lo sviluppo effettivo del software
- Tempo aggiuntivo per testing e verifica funzionalità

Lo sviluppo effettivo del software, contro intuitivamente, risulta essere il **processo più breve**, semplice e lineare dell'intero progetto.

Il motivo di ciò è semplice: durante le fasi di analisi e progettazione sono state effettuate **tutte le scelte necessarie allo sviluppo**, coprendo qualsiasi ambito. Di conseguenza, una volta conclusa la progettazione, lo sviluppo del software risulta essere un semplice lavoro manuale richiedente solo la conversione da funzionalità concettuali a codice applicativo.

Definition 1. Attori del progetto

Nell'ambito della progettazione del software, definiamo come **attori** qualsiasi ente e/o persona coinvolta nel progetto. In particolare, identifichiamo i seguenti attori:

- Committente
- Esperto del dominio
- Analista
- Progettista
- Programmatore
- Utente finale
- Manutentore

Un ente/persona può ricoprire **uno o più ruoli**. Ad esempio, il committente potrebbe essere anche l'utente finale del programma.

Esempio:

- Il Comune di XYZ intende automatizzare la gestione delle informazioni relative alle contravvenzioni elevate sul suo territorio. In particolare, intende dotare ogni vigile di una app per smartphone che gli consenta di comunicare al sistema informatico il veicolo a cui è stata comminata la contravvenzione, il luogo in cui è stata elevata e la natura dell'infrazione.

  Il sistema informatico provvederà a notificare, tramite posta ordinaria, la contravvenzione al cittadino interessato. Il Comune bandisce una gara per la realizzazione e manutenzione del sistema, che viene vinta dalla ditta ABC.

- In questo caso, gli attori coinvolti sono:
  - Committente: il comune di XYZ
  - Esperto del dominio: un funzionario del comune (o un altro professionista) esperto del codice della strada
  - Utenti finali: i vigili del comune
  - Analisti, progettisti, programmatori e manutentori: personale della ditta ABC commissionata

1.2 Ciclo di vita del software

All'interno del ciclo di vita di un software, identifichiamo **5 fasi principali**:

1. **Studio di fattibilità e raccolta dei requisiti**, ossia la valutazione dei costi e dei benefici, la pianificazione delle attività e delle risorse del progetto, l'individuazione dell'ambiente di programmazione e la raccolta dei **requisiti** per la realizzazione
2. **Analisi dei requisiti**, ossia l'analisi delle funzionalità del software, descrivendone il dominio e specificando le funzioni di ogni componente attraverso uno **schema concettuale**
3. **Progettazione e realizzazione**, ossia la definizione delle metodologie di realizzazione delle funzionalità individuate, definendo l'architettura del programma, scrivendo e documentando il codice effettivo
4. **Verifica e Testing**, ossia la verifica del corretto funzionamento del software
5. **Manutenzione**, ossia la correzione e l'aggiornamento del software

Idealmente, tali fasi dovrebbero essere realizzate seguendo un **modello a cascata**, ossia una successiva all'altra, senza mai tornare indietro. Tuttavia, ciò risulta impossibile, poiché durante ogni fase potrebbero insorgere eventuali problematiche o riprogettazioni dovute a migliorie o cambiamenti di idea del committente.

Nella pratica, quindi, viene utilizzato un **modello a spirale**, dove ogni fase viene realizzata poco per volta, passando il poco lavoro fatto alla fase successiva, la quale procederà analogamente. In tal modo, al termine di una "prima versione" di ogni fase, è possibile procedere già con la "seconda versione", la quale sarà più estensiva della precedente, e così via.

1.2.1 Analisi dei requisiti

Durante l'analisi dei requisiti, gli analisti si occupano di cogliere le implicazioni di ogni singolo requisito, specificandoli il più possibile con l'obiettivo di **formalizzarli**, eliminando **incompletezze**, **inconsistenze** ed **ambiguità**, fornendo una **specifica delle funzionalità** da realizzare ed elaborando uno **schema concettuale** che sarà di riferimento per tutte le successive fasi del ciclo di vita del software.

In particolare, lo schema concettuale è composto da:

- Diagrammi realizzati con opportuni linguaggi grafici di modellazione, ad esempio il linguaggio **Unified Modeling Language (UML)**
- Documenti di specifica

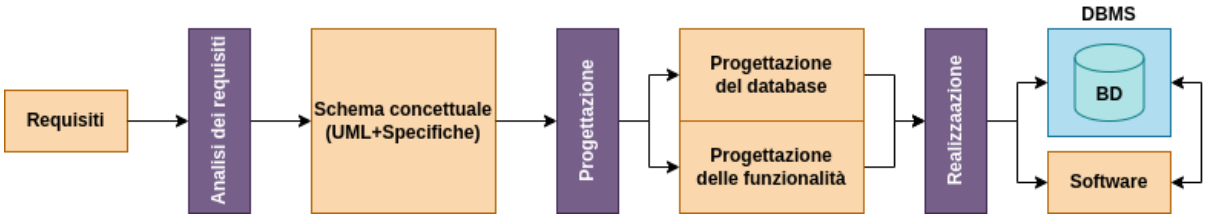
Tramite tali componenti, lo schema concettuale è in grado di descrivere completamente e precisamente il sistema da realizzare secondo diverse prospettive (quali sono e come organizzare i dati di interesse, le funzioanalità, ...)

Il **linguaggio UML** fornisce costrutti per modellare la gran parte degli aspetti, sia statici che dinamici, utilizzando un approccio **orientato agli oggetti**:

- Gli oggetti di interesse (ossia i dati) vengono organizzati secondo gerarchie di classi, descritte tramite un **diagramma delle classi**
- L'evoluzione possibile degli oggetti nel tempo viene descritta attraverso un **diagramma di transizione degli stati**
- Le funzionalità offerte dal sistema e i relativi attori che possono utilizzarle vengono descritte tramite un **diagramma degli use-case**
- Le interazioni tra gli oggetti e i processi del sistema vengono descritti in **diagrammi di sequenza, collaborazione ed attività**
- L'architettura del sistema viene progettata tramite **diagrammi di componenti e di deployment**

Nel caso della progettazione di una base di dati, l'approccio progettuale segue come:

1. Analisi tramite UML e documenti di specifica per modellare il sistema sotto le varie prospettive, decidendo quali siano i dati di interesse e la struttura di essi
2. Progettazione delle metodologie di memorizzazione di tali dati (ad esempio la scelta di un DBMS)



Dunque, nel linguaggio UML l'analisi degli aspetti relativi ai dati confluisce nel **diagramma delle classi**, il quale tuttavia contiene anche informazioni riguardo le operazioni svolte su tali dati, rendendo alcuni vincoli sui dati non sono esprimibili a pieno.

Di conseguenza, nella fase di analisi vengono utilizzati linguaggi di modellazione precursori dell'UML, orientati **esplicitamente a modellare solo i dati**. Il linguaggi più diffuso per tale scopo è il **linguaggi Entity-Relationship (ER)**.

Capitolo 2

Analisi concettuale dei requisiti

2.1 Linguaggio Entity-Relationship

Definition 2. Livello intensionale ed estensionale

Nella logica matematica, un oggetto, concetto o termine può essere descritto a:

- **Livello intensionale**, ossia specificandone le caratteristiche generali rispettate da ogni oggetto a cui il termine è associato
- **Livello estensionale**, ossia specificandone le caratteristiche specifiche di un oggetto, le quali lo differenziano da altri oggetti a cui il termine è associato

Esempio:

- Per riassumere tale concetto, nell'uso delle classi Java avremmo:

```
// Livello intensionale
public class Person {
    String name;
    String surname;
    public Persona(String n, String c){
        ...
    }
    ...
}

public static void main(String[] args){
    // Livello estensionale
    Person mario = new Person("Mario", "Rossi);
    ...
}
```

Definition 3. Linguaggio Entity-Relationship

Il **linguaggio Entity-Relationship (ER)** è un linguaggio grafico atto alla modellazione dei dati di interesse di un'applicazione, fornendo costrutti a cui è associata una sintassi ed una sematica di utilizzo.

Un **diagramma ER** descrive i dati solo a **livello intensionale**, ossia la **struttura** che essi assumono (dunque non al livello estensionale ossia i dati assunti da ogni istanza dell'oggetto, i quali possono variare nel tempo).

2.1.1 Entity e Domini

Definition 4. Entity

Un'**entity** rappresenta una **classe di oggetti** (ossia persone, cose, fatti, ...) di interesse per il dominio applicativo. Ogni istanza di un'entity possiede **proprietà comuni** (livello intensionale) ed hanno **esistenza autonoma** rispetto alle altre istanze (livello estensionale).

Definition 5. Attributi di un'entity

Un'entity può possedere degli **attributi**, ossia delle **proprietà locali**, i quali associano un valore ad un certo **dominio** (o tipo). I domini dei vari attributi vengono descritti all'interno di un **dizionario dei dati** esterno al diagramma ER.

Proposition 1. Domini di attributi

Principalmente, all'interno del linguaggio ER vengono utilizzati dei **domini elementari**, ossia **Stringa**, **Intero**, **Reale**, **Data**, **Ora**, **Dataora**.

Ognuno di tali domini può essere **ristretto** tramite dei **vincoli di dominio** (ad esempio, "intero > 0" corrisponde ad un attributo di tipo intero positivo).

Ulteriori domini utilizzato nel linguaggio ER sono:

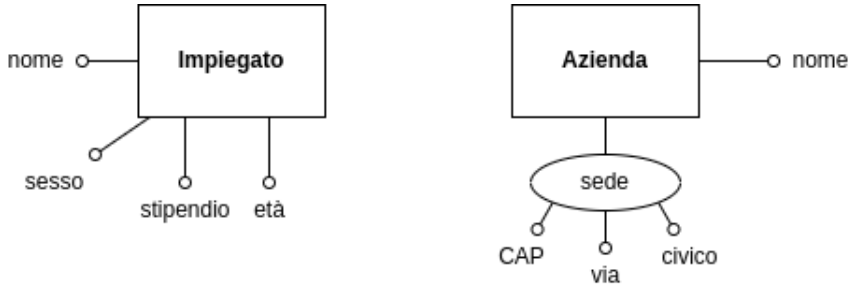
- **Dominio enumerativo**, il quale viene artificialmente deciso dall'analista e viene indicato come un **insieme di simboli** (ad esempio, {uomo, donna})
- **Dominio record**, il quale corrisponde ad una tupla di domini. Viene utilizzato per gli **attributi composti**.

Esempio:

- Consideriamo le seguenti due entity:
  - L'entity **Impiegato** possiede gli attributi **nome** (stringa), **sesso** (enum di tipo uomo o donna), **stipendio** (intero positivo), età (intero positivo)

  - L'entity **Azienda** possiede gli attibuti **nome** (stringa), **sede** (**record** composto da CAP (intero positivo), via (stringa) e numero civico(intero positivo))

- In linguaggio ER, rappresenteremmo tali entity e tali attributi nel seguente modo:

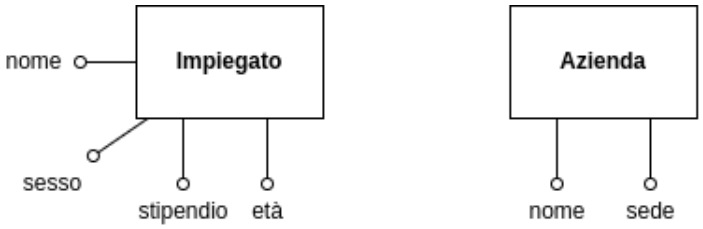


  mentre il relativo dizionario dei dati corrisponderebbe a:

Impiegato	
Nome	Stringa
Sesso	{Uomo, Donna}
Età	Intero > 0
Stipendio	Intero > 0

Azienda	
Nome	Stringa
Sede	Record(
	CAP: Intero >0,
	via: Stringa,
	civico: Intero >0)

- Ad ogni istanza di Impiegato, dunque, verrà associato uno ed un solo valore di tipo stringa per l'attributo nome, uno ed un solo valore di tipo intero positivo per l'attributo età ed uno ed un solo valore di tipo intero positivo per l'attributo stipendio.
- Per comodità, i campi appartenenti ad un attributo composto vengono **omessi** dal diagramma ER, poiché essi sono già indicati nel dizionario dei dati:



- Consideriamo quindi le seguenti due istanze di Impiegato:
  - Nome: "Anna", sesso: "Donna", età: 35 e stipendio: 40000
  - Nome: "Anna", sesso: "Donna", età: 35 e stipendio: 40000

  Poiché ogni istanza di un'entity possiede **esistenza autonoma**, tali istanze, seppur coincidenti nel valore di ogni attributo, sono **distinte tra loro**. Dunque, esse possono coesistere.

2.1.2 Relationship

Definition 6. Relationship

Una **relationship** esprime un possibile legame tra istanze di due o più entity. Il numero di entity coinvolte nella relationship viene detto **grado** o **arità**.

A livello estensionale, una relationship r tra due entity E ed F corrisponde ad una **relazione matematica**, ossia un sottoinsieme del prodotto cartesiano tra le due entity

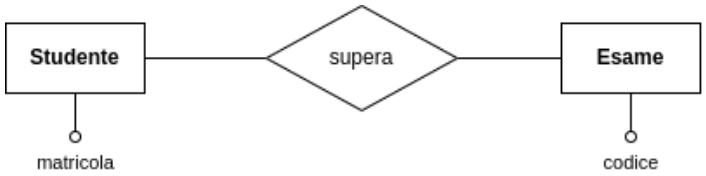
$$r \subseteq E \times F$$

Observation 1

Sia r una relationship tra le entity E ed F. Poiché una relazione matematica è un insieme, ne segue che non possano esistere in r **due istanze** che legano la **stessa coppia di istanze** di E ed F

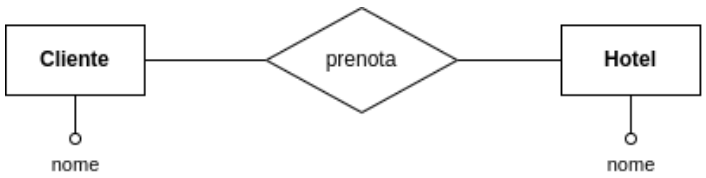
Esempi:

1. Consideriamo la seguente relationship tra le entity Studente ed Esame:



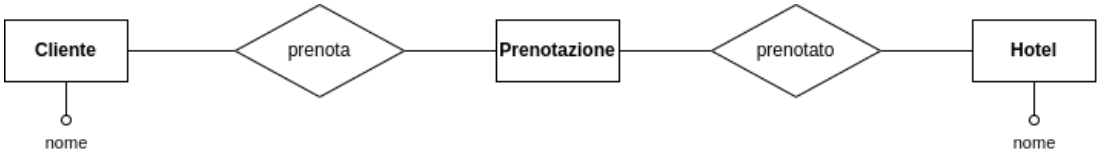
   In tal caso, il vincolo di inesistenza di duplicati all'interno della relationship risulta essere **adeguato**, poiché ciò impedisce che lo stesso studente possa sostenere più volte lo stesso esame

2. Consideriamo la seguente relationship tra le entity Cliente ed Hotel:



   In tal caso, il vincolo di inesistenza di duplicati all'interno della relationship risulta essere **inadatto**, poiché ciò impedisce che lo stesso cliente possa prenotare più volte lo stesso hotel.

   Per risolvere tale problema, è necessario creare un'ulteriore entity Prenotazione che faccia da "ponte" tra le due relationship.



Definition 7. Vincoli di integrità

Definiamo come **vincoli di integrità** delle **restrizioni** sulle entity e sulle relationship a livello estensionale

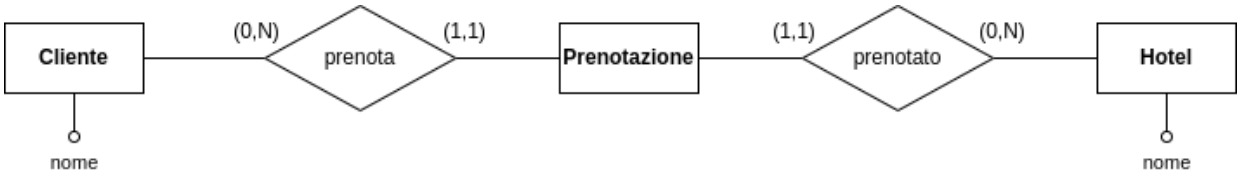
Definition 8. Vincolo di molteplicità

Definiamo come **vincolo di molteplicità** un vincolo di integrità che esprime il **numero volte** in cui un'istanza di un'entity può essere coinvolta in una relationship.

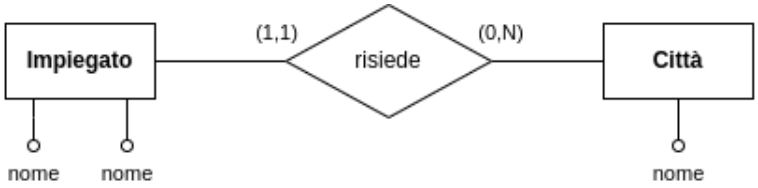
Tali vincoli vengono rappresentati con delle **coppie di interi**, rappresentanti il limite minimo e massimo della molteplicità di un'istanza dell'entity all'interno della relationship.

Esempio:

- Riprendendo l'esempio precedente inerente alle prenotazioni di hotel, i vincoli di molteplicità di tale diagramma ER corrispondono a:



- Consideriamo il seguente diagramma ER:



- In tale diagramma, il vincolo di molteplicità dell'entity Impiegato relativa alla relationship "risiede" risulta essere $(1, 1)$, indicante che ogni impiegato risiede in minimo una città e massimo una città (dunque ogni impiegato risiede in una città)
- Analogamente, il vincolo associato all'entity Città risulta essere $(0, N)$, indicante che in ogni città possa risiedere nessun impiegato o un numero indefinito di impiegati

Observation 2

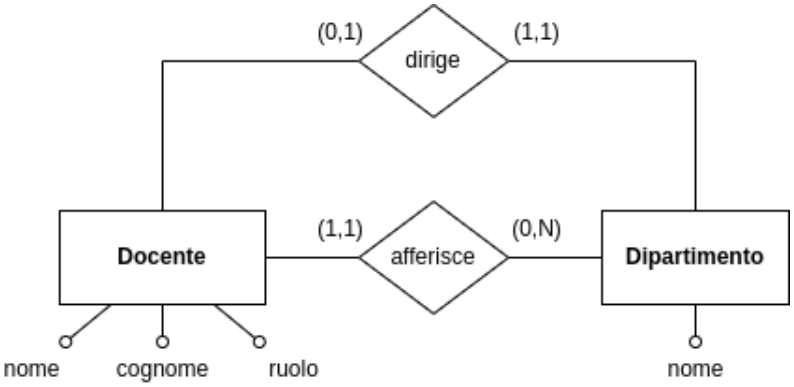
Due entity possono essere involte in **più relationship tra di esse**

Esempio:

- Vogliamo modellare i docenti di un ateneo. Di ogni docente vogliamo mantenere, il nome, il cognome, il ruolo assunto (il quale può essere RU, ossia ricercatore universitario, PA, ossia professore associato, e PO, ossia professore ordinario) e

di dipartimento a cui afferisce. Inoltre, vogliamo tenere traccia del nome di ogni dipartimento e del suo direttore, corrispondente ad un docente.

- Il diagramma ER corrispondente sarà:



Definition 9. Ruoli di relationship

Nel caso in cui un'entity sia **involta più volte in una relationship**, per evitare ambiguità vengono assegnati dei **ruoli** assunti dalle possibili istanze di tale entity all'interno della relationship stessa, i quali vengono indicati all'interno delle istanze della relationship con un'**etichetta** indicandone il ruolo.

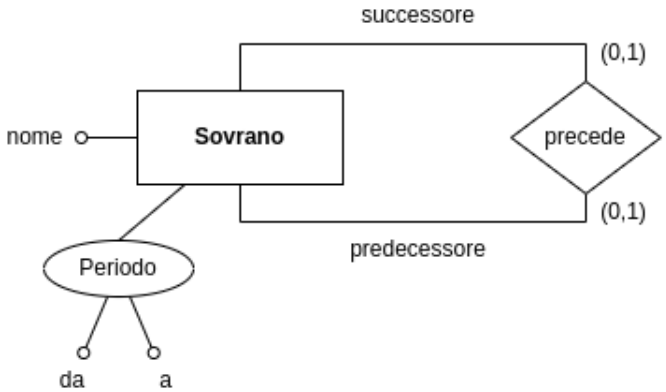
Nel caso in cui l'entity sia involta una sola volta nella relationship, le verrà assegnato automaticamente un ruolo corrispondente al suo nome.

Observation 3

Il ruolo di un'entity può essere specificato anche in casi in cui non sia obbligatorio, rendendo più esplicito il suo compito all'interno della relationship

Esempi:

- Nel seguente diagramma ER, l'entity Sovrano è involta due volte nella relationship "precede"

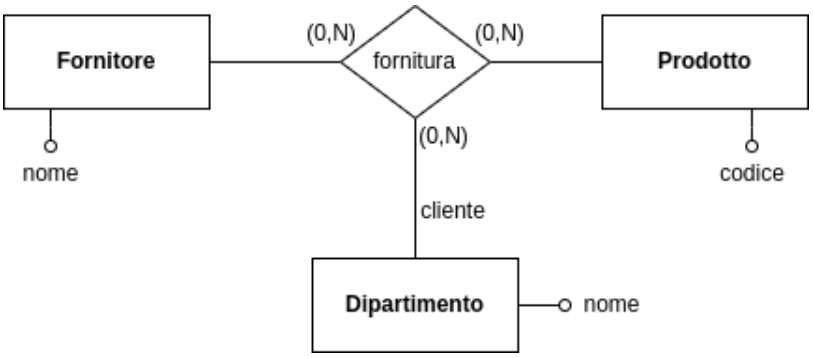


- In tal caso, le tuple rappresentanti le istanze della relationship avranno la seguente forma:

  $$(\text{predecessore: } s_1 \text{ , successore: } s_2 \text{ })$$

  dove $s_1, s_2 \in$ Sovrano

- Nel seguente diagramma ER, viene assegnato il ruolo "cliente" all'entity Dipartimento, in modo da esplicitare il suo compito all'interno della relationship "fornitura"



Proposition 2. Semantica di una relationship

A livello estensionale, una **relationship** r tra le entity $E_1, \ldots, E_n$, aventi rispettivi ruoli $u_1, \ldots, u_n$, corrisponde ad un insieme di $n-uple$ nella forma:

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

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

Nota: le entity non devono essere necessariamente tutte distinte

Definition 10. Attributi di una relationship

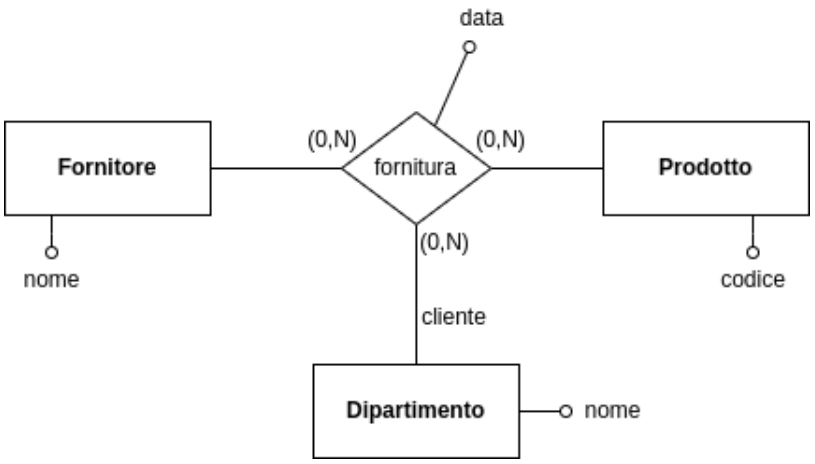
Una relationship può possedere degli **attributi**, ossia delle **proprietà locali**, i quali associano ad ogni istanza della relationship un valore in un certo dominio.

Observation 4

Associare un attributo ad una relationship **non ne modifica la struttura**. Dunque, due istanze relative alle stesse istanze di due entity ma con valore diverso per un attributo non possono coesistere

Esempi:

1. - Riprendiamo il precedente esempio inerente alle forniture effettuate ad un dipartimento, modificandolo al fine di creare uno storico delle forniture effettuate.
   - Come prima soluzione, aggiungiamo un attributo "data" alla relationship fornitura.



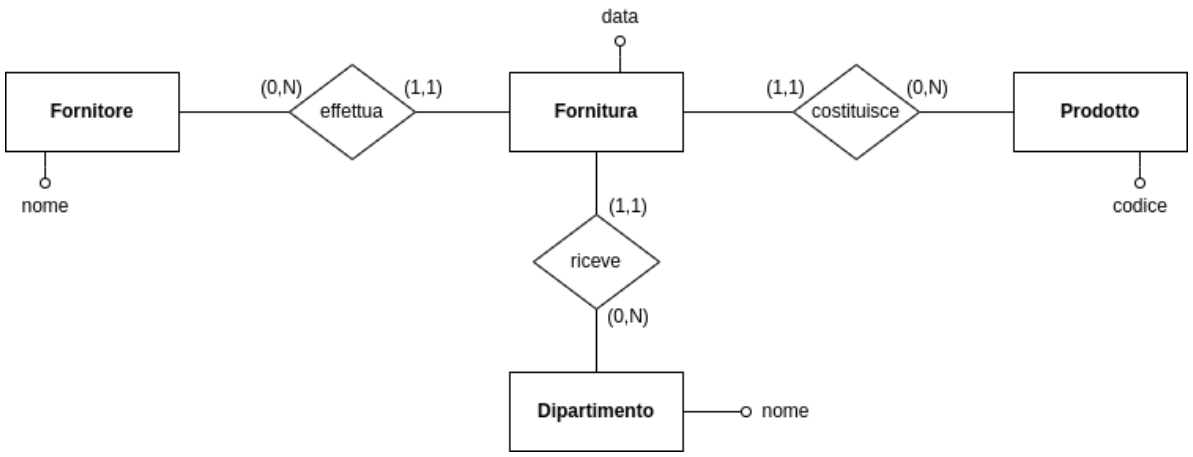
- Tuttavia, tale soluzione risulta essere **errata**: le tiple di tale relationship saranno comunque nella forma

  Fornitore: f, cliente: d, Prodotto:p

  e non

  Fornitore: f, cliente: d, Prodotto:p, Data:data

- Di conseguenza, le triple (AziendaXYZ, Dip3, Cartone, 12/02/13) e (AziendaXYZ, Dip3, Cartone, 25/02/13) fanno riferimento alla stessa istanza, ossia l'istanza (AziendaXYZ, Dip3, Cartone), dunque **non possono coesistere**.
- Per realizzare lo storico delle forniture effettutate, dunque, è necessaria un'ulteriore entity Fornitura possedente un proprio attributo "data":



2. - Si vuole sviluppare un sistema informativo per la gestione dei dati sul personale di una certa azienda costituita da diversi dipartimenti. I dati di interesse per il sistema sono impiegati, dipartimenti, direttori dei dipartimenti e progetti aziendali.

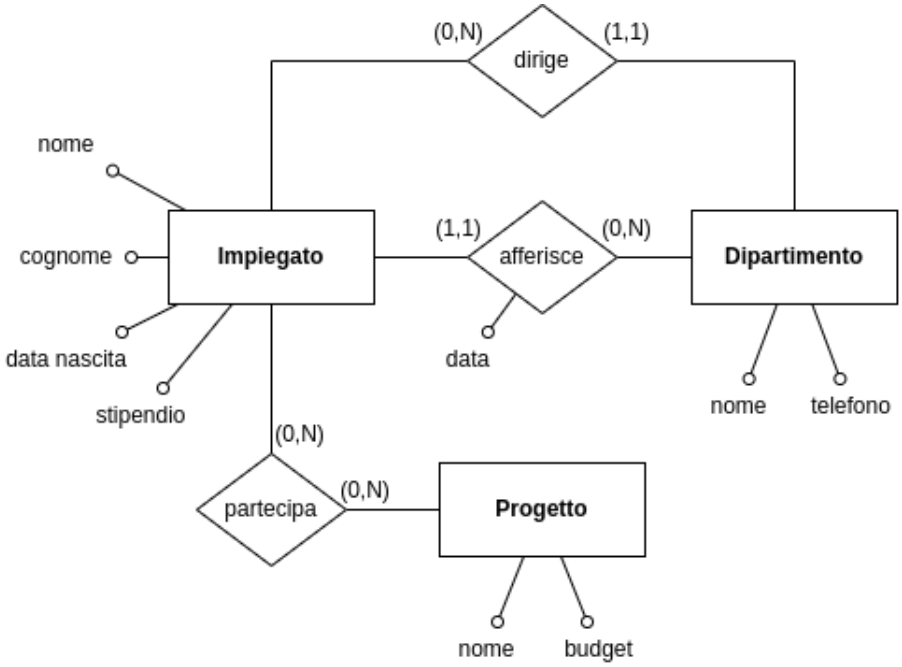
     Di ogni impiegato interessa conoscere il nome, il cognome, la data di nascita e lo stipendio attuale, il dipartimento (esattamente uno) al quale afferisce.

     Di ogni dipartimento interessa conoscere il nome, il numero di telefono del centralino, e la data di afferenza di ognuno degli impiegati che vi lavorano.

Di ogni dipartimento interessa conoscere inoltre il direttore, corrispondente ad uno degli impiegati dell'azienda.

Il sistema deve permettere di rappresentare i progetti aziendali nei quali sono coinvolti i diversi impiegati. Di ogni progetto interessa il nome ed il budget. Ogni impiegato può partecipare ad un numero qualsiasi di progetti.

- Il diagramma ER corrisponde a:



- Il dizionario dei dati corrisponde a:

Entity Impiegato	
Nome	Stringa
Cognome	Stringa
Data Nascita	Data
Stipendio	Reale > 0

Entity Dipartimento	
Nome	Stringa
Telefono	Stringa

Entity Progetto	
Nome	Stringa
Budget	Reale > 0

Rel. dirige
Impiegato (0,N) – (1,1) Dipartimento

Rel. afferisce
Impiegato (1,1) – (0,N) Dipartimento
Data Data

Rel. partecipa
Impiegato (0,N) – (0,N) Progetto

2.1.3 Ereditarietà e Generalizzazioni tra entity

Definition 11. Relazione is-a tra entity

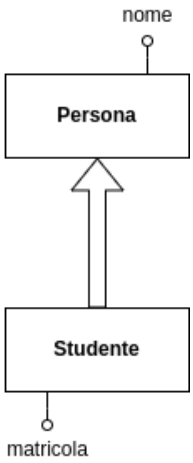
Definiamo come **relazione is-a tra due entity** E **ed** F una relazione indicante che F sia un'entity derivata da E, **ereditandone** tutte le istanze, estendendole con aggiuntivi attributi e relationship.

A livello estensionale, l'insieme delle istanze di F è un **sottoinsieme** dell'insieme delle istanze di E.

Inoltre, definiamo E come **entity base** e F come **entity derivata**

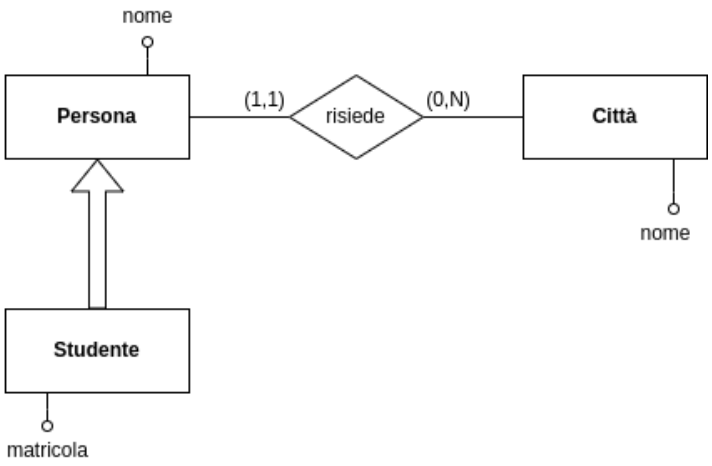
Esempio:

- Consideriamo il seguente diagramma ER:

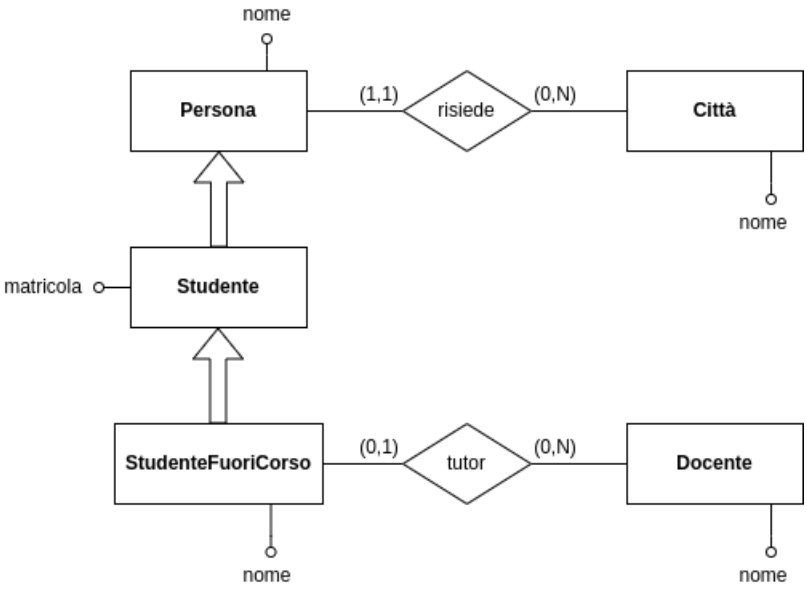


  In tal caso, ogni istanza di Studente è anche un'istanza di Persona, dunque ogni studente possiede anche un nome, oltre alla matricola aggiuntiva. Ovviamente, il viceversa non vale, dunque una persona non è detto che sia uno studente.

- Nel caso in cui l'entity Persona sia in una relationship con un altra entity, anche l'entity Studente sarà involta in tale relationship



- L'entity derivata Studente può a sua volta essere base di altre entity. Inoltre, nel caso in cui le entity derivate da Studente siano in una relationship con altre entity, l'entity Studente **non** apparterrà a tale relationship

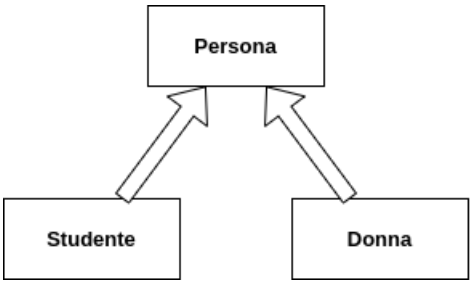


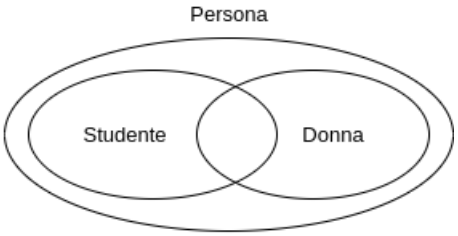
Observation 5. Derivazione multipla

Un'entity può essere **base per più entity derivate**, le quali possono avere istanze in comune

Esempio:

- Nel seguente diagramma ER, può esistere un'istanza di Studente che è anche istanza di Donna





Observation 6. Ereditarietà singola

Un'entity non può essere derivata da **più di una entity base**

Esempio:

- Un'entity StudenteLavoratore non può essere derivata sia da un'entity Studente sia da un'entity Lavoratore

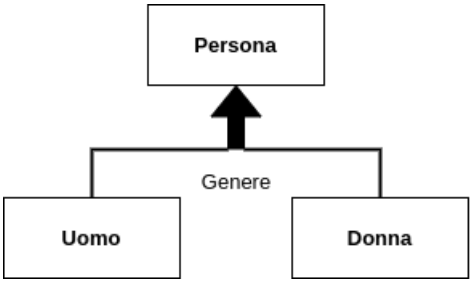
Definition 12. Generalizzazione completa e non completa

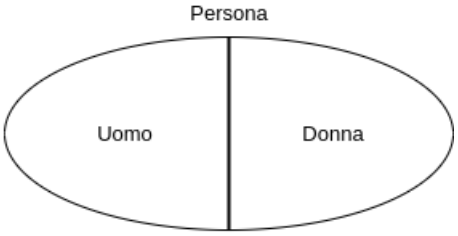
Definiamo come **generalizzazione** un particolare tipo di **relazione is-a** dove le entity derivate dalla base sono **disgiunte** tra loro.

Una generalizzazione viene detta **completa** se su di essa vale il **vincolo di completezza**, ossia non possono esistere istanze dell'entity base non ricadenti in una delle entity derivate. In caso contrario, una generalizzazione viene detta **non completa**.

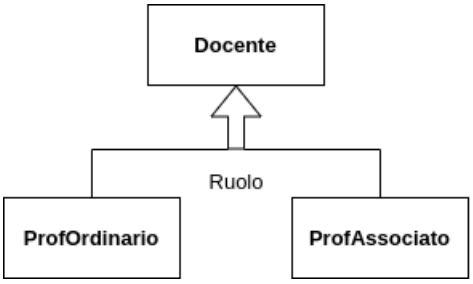
Esempi:

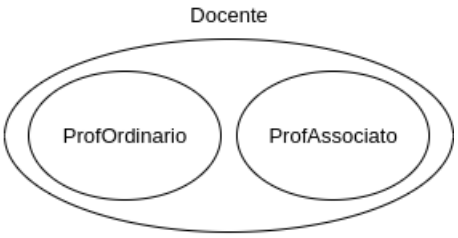
- Nel seguente diagramma ER, non può esistere un'istanza di Persona che non sia un'istanza di Uomo o un'istanza di Donna (generalizzazione completa)



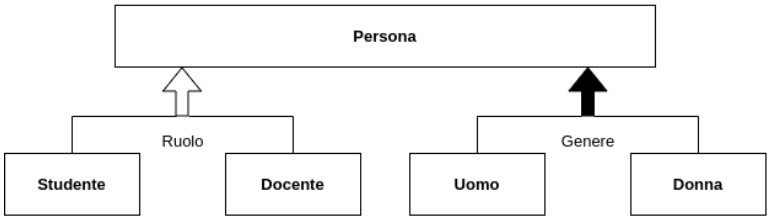


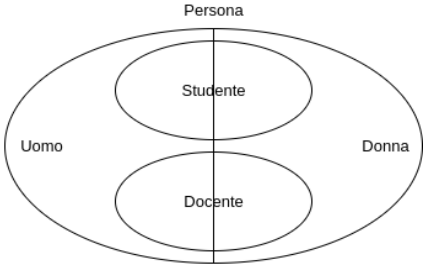
- Nel seguente diagramma ER, può esistere un'istanza di Docente che non sia un'istanza di ProfOrdinario o un'istanza di ProfAssociato (generalizzazione non completa)





- Nel seguente diagramma ER, possono esistere:
  - Istanze di Persone Uomini oppure Donne
  - Istanze di Studenti Uomini oppure Donne
  - Istanze di Docenti Uomini oppure Donne





2.1.4 Ereditarietà tra relationship

Definition 13. Relazione is-a tra relationship

Definiamo come **relazione is-a tra due relationship** r **ed** q una relazione indicante che q sia una relationship derivata da r, **ereditandone** tutte leistanze, estendendole con aggiuntivi attributi e relationship.

A livello estensionale, l'insieme delle istanze di q è un **sottoinsieme** dell'insieme delle istanze di r.

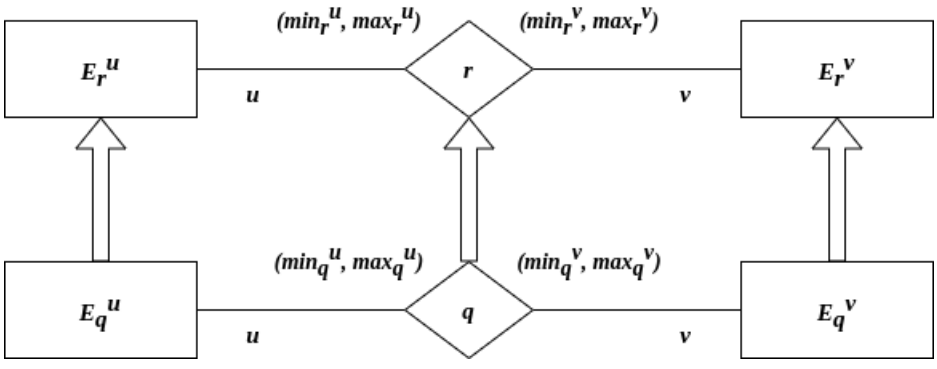
Inoltre, definiamo r come **relationship base** e q come **relationship derivata**

Proposition 3. Condizioni necessarie per relazioni is-a tra relationship

Siano u e v i ruoli assunti dalle entity E_r^u ed E_r^v coinvolte nella relationship r.

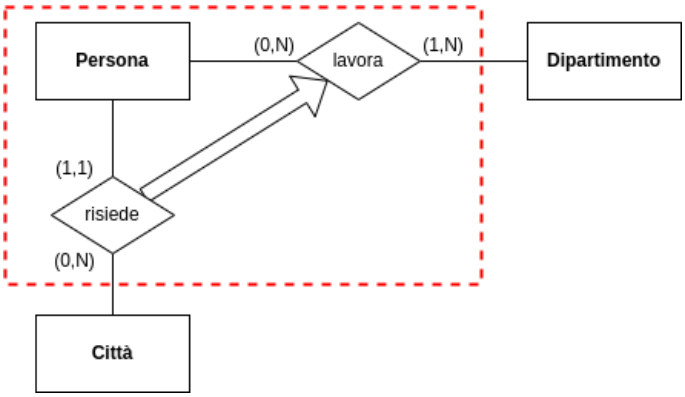
Data una **relationship** q **derivata da** r, affinché tale relazione is-a **possa esistere**, è necessario che:

- Le relationship r e q abbiano la **stessa arità**
- Le entity E_q^u e E_q^v coinvolte nella relationship q assumano rispettivamente gli **stessi ruoli** u e r
- E_q^u deve essere un'**entity derivata** da E_r^u **oppure** $E_q^u = E_r^u$
- E_q^v deve essere un'**entity derivata** da E_r^v **oppure** $E_q^v = E_r^v$
- Per le **molteplicità** (min_q^u, max_q^u) e (min_r^u, max_r^u) del ruolo u rispettivamente in q e in r, deve valere:
  - $max_q^u \leq max_r^u$
  - $E_q^u = E_r^u \implies min_q^u \leq min_r^u$
- Per le **molteplicità** (min_q^v, max_q^v) e (min_r^v, max_r^v) del ruolo v rispettivamente in q e in r, deve valere:
  - $max_q^v \leq max_r^v$
  - $E_q^v = E_r^v \implies min_q^v \leq min_r^v$

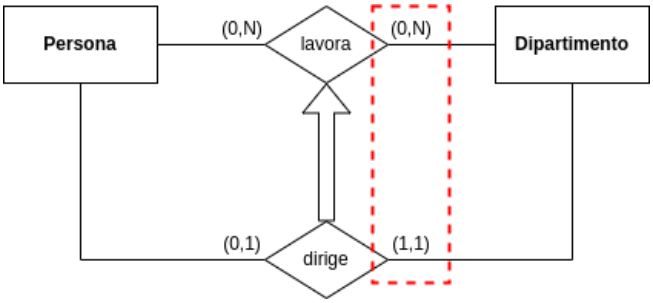


Esempi:

1. • Consideriamo il seguente diagramma ER. Notiamo facilmente come la realtà modellata violi le condizioni di esistenza della relazione is-a tra le relationship "risiede" e "lavora", poiché tali relationship sono chiaramente non dello stesso tipo, modellando una realtà non avente alcun senso logico.

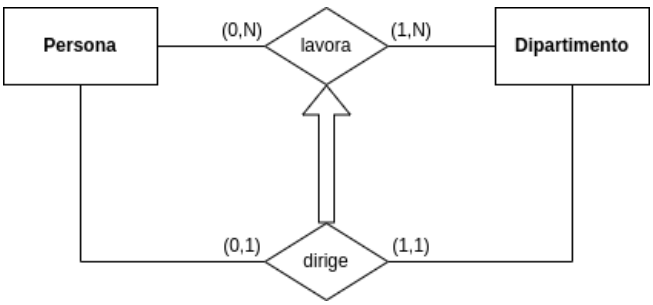


2. • Consideriamo il seguente diagramma ER. La realtà modellata è la seguente:
   - Ogni Persona può lavorare in nessuno o più dipartimenti
   - In ogni Dipartimento possono lavorare nessuna o più persone
   - Ogni Persona può dirigere nessuno o il dipartimento in cui lavora
   - Ogni Dipartimento deve essere diretto da una persona che vi lavora

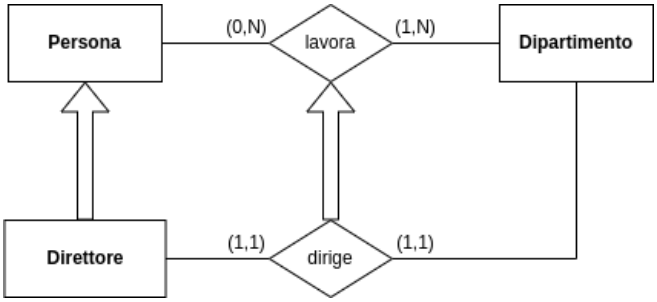


   • Sebbene la relazione is-a tra le relationship "dirige" e "lavora" rispetti le condizioni di esistenza, essa viola le condizioni di corretta modellazione: ogni Dipartimento deve avere un direttore che vi lavora, ma contemporaneamente in ogni Dipartimento potrebbe anche non lavorare alcuna persona.

   • Restingendo il vincolo di molteplicità del ruolo dipartimento nella relationship "lavora", tale incongruenza logica viene risolta



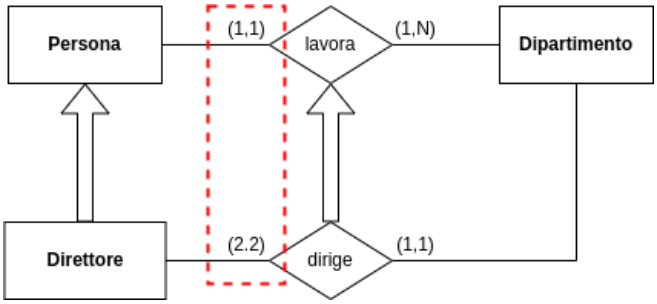
3. • Consideriamo il seguente diagramma ER. La realtà modellata è la seguente:
   - Ogni Persona può lavorare in nessuno o più dipartimenti
   - In ogni Dipartimento possono lavorare una o più Persone
   - Ogni Direttore dirige il Dipartimento in cui lavora
   - Ogni Dipartimento è diretto da un Direttore che vi lavora



   • A differenza dell'esempio precedente, in tal caso non vengono violate le condizioni di corretta modellazione, poiché le istanze dell'entity Direttore, nonostante esse siano un sottoinsieme delle istanze di Persona, esistono indipendentemente da quest'ultime.

   Di conseguenza, non va a crearsi alcuna incongruenza logica: possono esistere delle istanze di Persona che non lavorano in alcun Dipartimento, ma ogni istanza di Direttore lavora nel dipartimento che dirige

4. • Consideriamo il seguente diagramma ER. La realtà modellata è la seguente:
   - Ogni Persona lavora in un Dipartimento
   - In ogni Dipartimento lavorano una o più Persone
   - Ogni Dipartimento è diretto da un Direttore che vi lavora
   - Ogni Direttore dirige due Dipartimenti in cui lavora



   • Anche in tal caso, è presente un'errore di pessima modellazione dovuto ad un'incongruenza logica: ogni Direttore deve dirigere due Dipartimenti in cui lavora, tuttavia, poiché ogni Direttore è anche una Persona, esso può lavorare massimo in un Dipartimento.

   Di conseguenza, per via di tale problematica, non possono esistere istanze di Direttore fatta eccezione dell'istanza vuota

2.1.5 Vincoli di cardinalità, identificazione ed esterni

Definition 14. Vincolo di cardinalità su un attributo

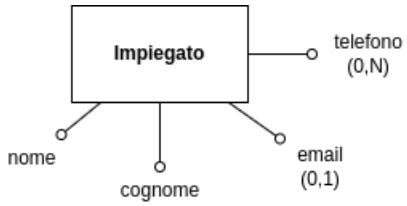
Definiamo come **vincolo di cardinalità su un attributo** un vincolo di integrità che esprime il **numero di valori** assunti dall'attributo designato di ogni istanza di un'entity.

Tali vincoli vengono rappresentati con delle **coppie di interi**, rappresentanti il limite minimo e massimo della cardinalità dell'attributo.

Se tale coppia non è indicata, viene assunto che range del vincolo sia $(1, 1)$

Esempio:

- Nel seguente diagramma ER, ogni istanza dell'entity Impiegato può possere nessuna o una sola email, nessun o più numeri di telefono e deve possenedere obbligatoriamente un nome ed un cognome



Definition 15. Vincolo di identificazione

Definiamo come **vincolo di identificazione** un vincolo di integrità che definisce su un'entity un **identificatore**, ossia un insieme di attributi (di cardinalità $(1, 1)$) e/o ruoli di relationship in cui tale entity è coinvolta (di molteplicità $(1, 1)$) tale che non esistono due istanze dell'entity che coincidono in tutti i valori di tale insieme.

Tali vincoli vengono rappresentati da dei **pallini neri**

Observation 7

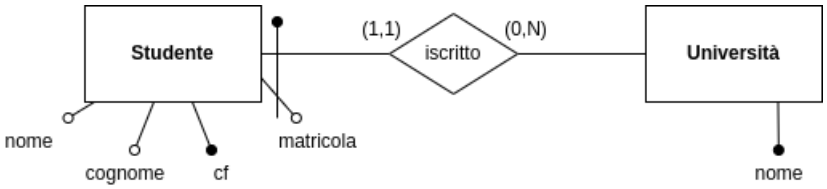
Gli identificatori utilizzati per un vincolo di identificazione devono essere **insiemi minimali di attributi**, ossia non deve poter esistere un sottoinsieme di attributi di tale identificatore che possa fungere a sua volta come identificatore

Esempi:

1. - Si vogliono rappresentare degli studenti, tenendo traccia di nome, cognome, matricola, codice fiscale e l'università di appartenenza, con il relativo nome dell'università.

     In particolare, vogliamo che non esistano due studenti con lo stesso codice fiscale, non esistano due università con lo stesso nome e che non esistano due studenti aventi la stessa matricola frequentanti la stessa università

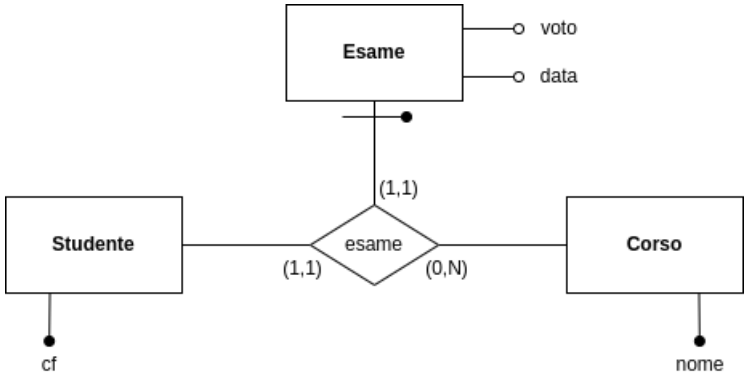
- Definiamo quindi tre identificatori:
  - Un identificatore `{cf}` sull'entity Studente
  - Un identificatore `{nome}` sull'entity Università
  - Un identificatore `{matricola, studente}`, dove matricola è un attributo dell'entity Studente e studente è il ruolo assunto da tale entity nella relationship "iscritto"
- Notiamo che, ad esempio, l'insieme `{cf, nome, cognome}` non è un identificatore minimale, poiché anche il sottoinsieme `{cf}` è in grado di identificare ogni istanza dell'entity Studente
- Il diagramma ER corrispondente sarà:



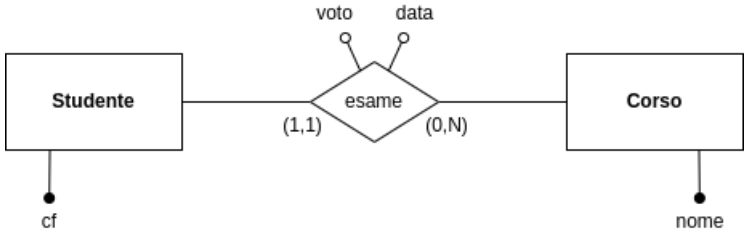
2. - Si vogliono rappresentare degli studenti (con relativo nome), dei corsi (con relativo nome) e gli esami sostenuti dagli studenti (con relativo voto e data)

     In particolare, vogliamo che non esistano due studenti con lo stesso codice fiscale, non esistano due corsi con lo stesso nome e che ogni verbalizzazione sia relativa ad uno ed un solo esame

   - Il diagramma ER corrispondente sarà:



   - Notiamo che, a livello estenzionale, il diagramma sottostante sia esattamente coincidente al precedente. Tuttavia, rendendo esplicita l'entity Esame è possibile involvere quest'ultima in ulteriori relationship (nel caso in cui vi fossero)



2.2 Diagramma UML degli use-case

Definition 16. Diagramma UML degli use-case

Il **diagramma UML degli use-case** rappresenta i vari **use-case** (o scenari di utilizzo) del sistema che si vuole modellare, dove ogni **use-case** racchiudente al suo interno un insieme omogeneo di **funzionalità** accedute ed accedibili da un gruppo omogeneo di **utenti** del sistema.

Ogni use-case viene rappresentato da un **nodo con titolo associato**.

Definition 17. Attore

Definiamo come **attore** il **ruolo assunto da un utente** (il quale può essere sia umano sia un sistema esterno) all'interno del sistema che si vuole modellare.

In particolare, **ogni utente** può essere rappresentato da **più attori** e **più utenti** possono essere rappresentati dallo **stesso attore**.

All'interno del diagramma UML degli use-case, ogni attore viene rappresentato da un **omino con nome associato**.

Definition 18. Associazione

All'interno del diagramma UML degli use-case, ogni attore del sistema viene **associato** ad uno use-case tramite un arco che li collega, modellando la **possibilità di accesso** da parte di tale attore alle **funzionalità** racchiuse in tale use-case.

Il nome di ogni associazione è, a meno di ambiguità, omissibile.

Esempio:

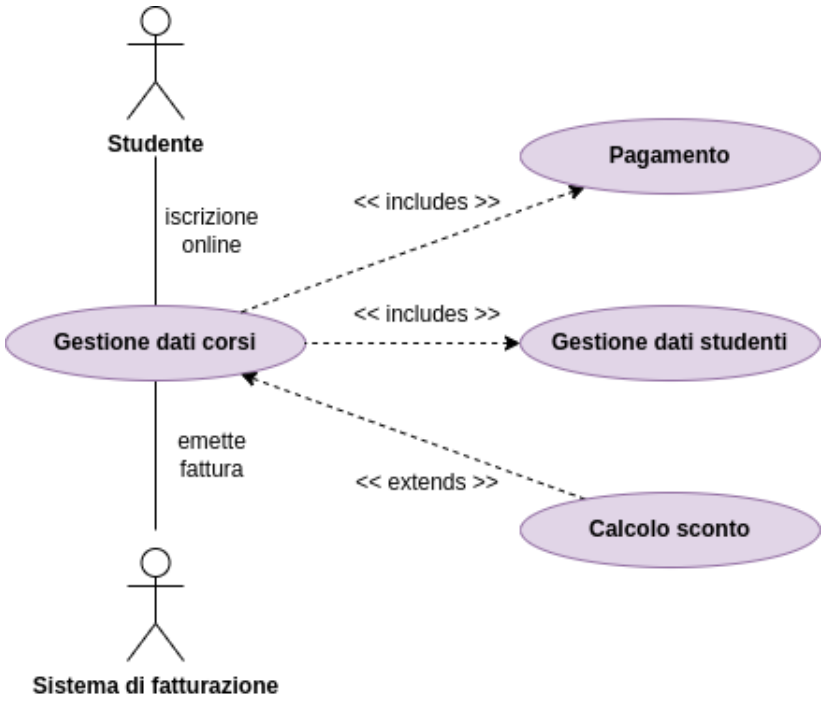
Definition 19. Inclusione ed Estensione

All'interno del diagramma UML degli use-case, ogni use-case del sistema può:

- **Includere** un altro use-case, modellando la **necessità** di tale use-case di usufruire di alcune di funzionalità presenti in tale use-case.
- **Estendere** un altro use-case, modellando la **possibilità** di concedere a quest'ultimo l'accesso ad alcune funzionalità racchiuse in tale use-case solo nel caso in cui si verifichino condizioni particolari.

Esempio:

- Nel seguente diagramma degli use-case, si ha che:
- – La gestione dei dati corsi include il pagamento e la gestione dei dati degli studenti
  – Il calcolo di uno sconto estende, in alcuni casi, la gestione dei corsi

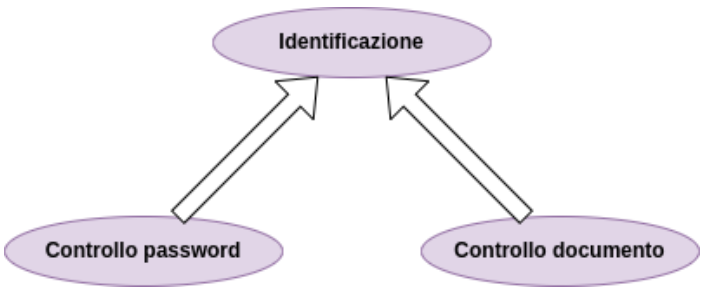


Definition 20. Generalizzazione tra use-case

Definiamo come **generalizzazione tra due use-case** una relazione indicante che uno use-case sia un **caso particolare di un altro use-case**, ereditandone tutte le funzioanalità ed estenendole con aggiuntive o modificando il comportamento di quelle ereditate. In particolare, tale relazione indica che uno use-case possa, in alcuni casi, **sostituire** lo use-case da cui eredita le funzionalità.

Esempio:

- Il comune processo di identificazione di un individuo può, a seconda dei casi, essere sostituito sia dal controllo di una password o di un documento



Definition 21. Generalizzazione tra attori

Definiamo come **generalizzazione tra due attori** una relazione indicante che un attore sia un **caso particolare di un altro attore**, **condividendone** tutti gli use-case associati a quest'ultimo, estenendoli con aggiuntivi.

Esempio:

- Riprendendo l'esempio precedente, uno Studente Convenzionato è un caso particolare attore del sistema involto negli stessi use-case dell'attore Studente