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

26 aprile 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 ri-progettazioni 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.

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

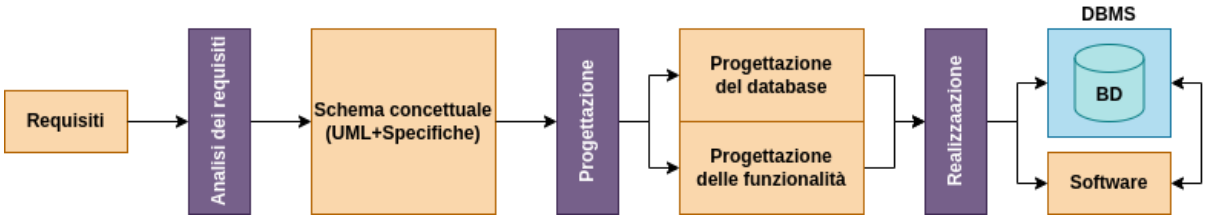
Tramite tali componenti, lo schema concettuale è in grado di descrivere completamente e precisamente il sistema da realizzare secondo diverse prospettive (quali sono e come organizzare i dati di interesse, le funzionalità, ...)

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 semantica 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:

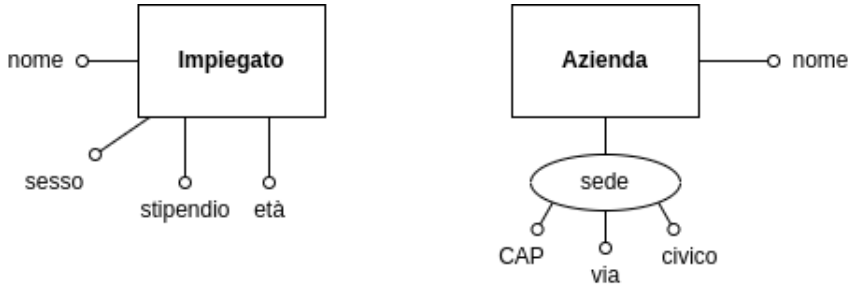
- **Dominio enumerativo**, il quale viene artificialmente deciso dall'analista e viene indicato come un **insieme di simboli** (ad esempio, {uomo, donna})
- **Dominio record**, il quale corrisponde ad una tupla di domini. Viene utilizzato per gli **attributi composti**.

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 attributi **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:

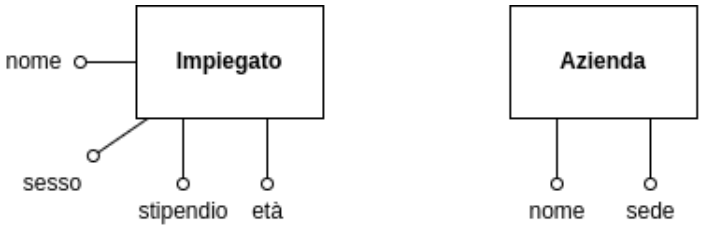


  mentre il relativo dizionario dei dati corrisponderebbe a:

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

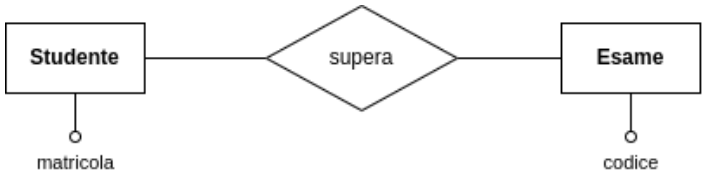
$$r \subseteq E \times F$$

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

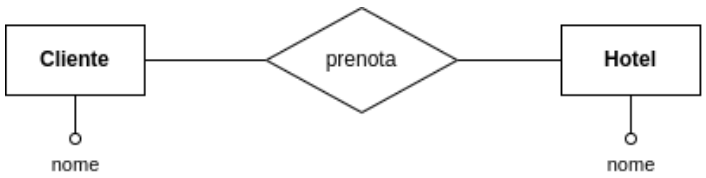
Esempi:

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



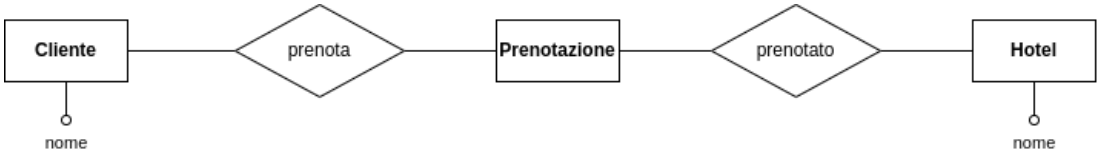
   In tal caso, il vincolo di inesistenza di duplicati all'interno della relationship risulta essere **adeguato**, poiché ciò impedisce che lo stesso studente possa sostenere più volte lo stesso esame

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

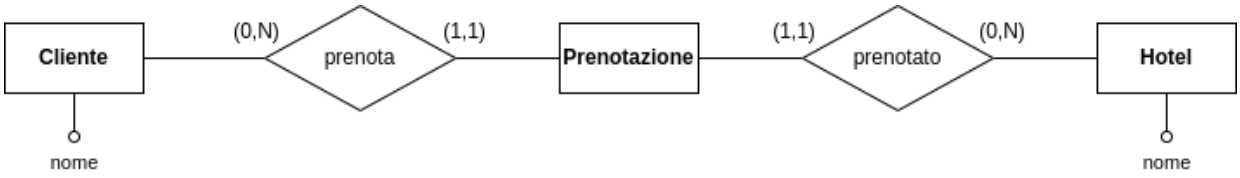
Definition 8. Vincolo di molteplicità

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.

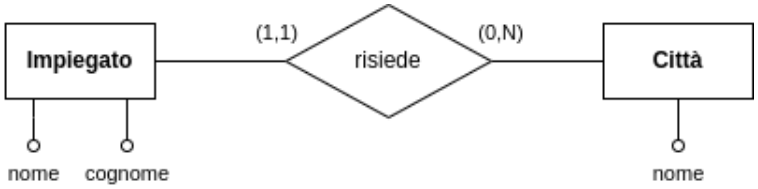
Tali vincoli vengono rappresentati con delle **coppie di interi**, rappresentanti il limite minimo e massimo della molteplicità di un'istanza dell'entity all'interno della relationship.

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

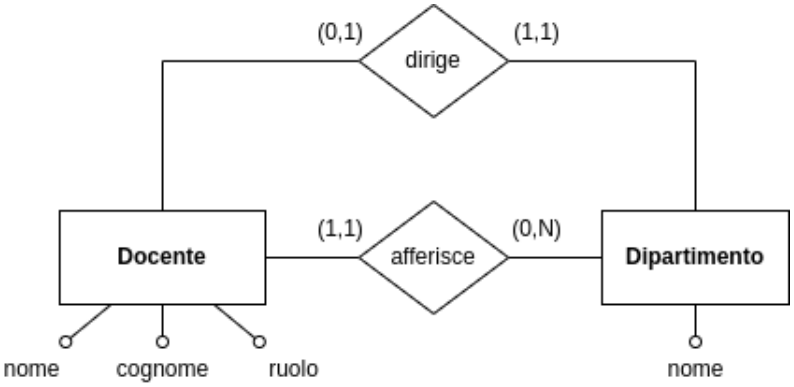
Due entity possono essere involte in **più relationship tra di esse**

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.

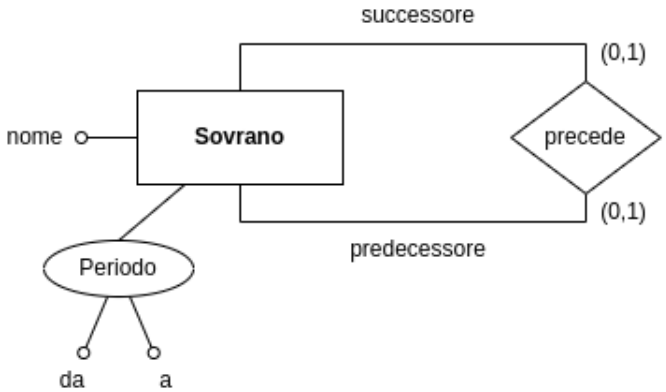
Nel caso in cui l'entity sia involta una sola volta nella relationship, le verrà assegnato automaticamente un ruolo corrispondente al suo nome.

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"

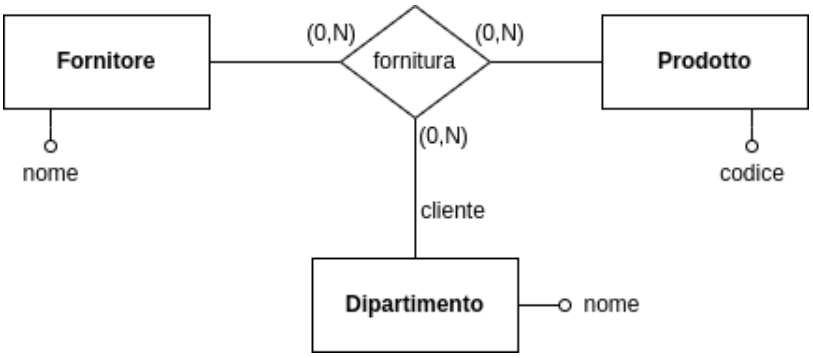


- In tal caso, le tuple rappresentanti le istanze della relationship avranno la seguente forma:

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

  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

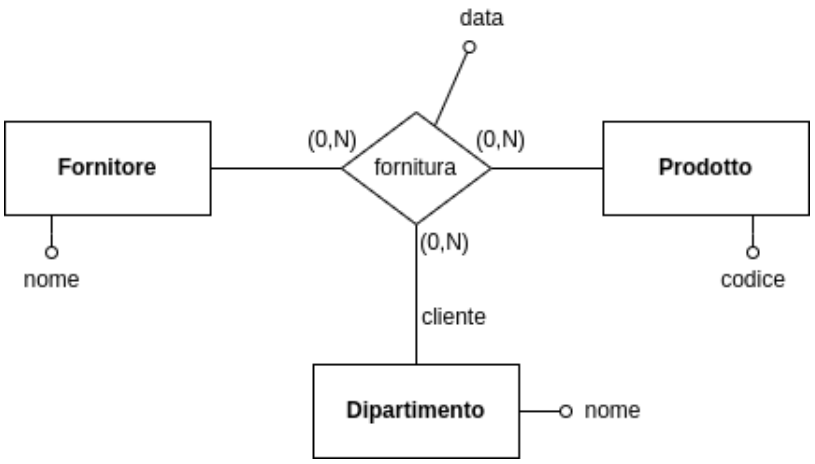
Una relationship può possedere degli **attributi**, ossia delle **proprietà locali**, i quali associano ad ogni istanza della relationship un valore in un certo dominio.

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.



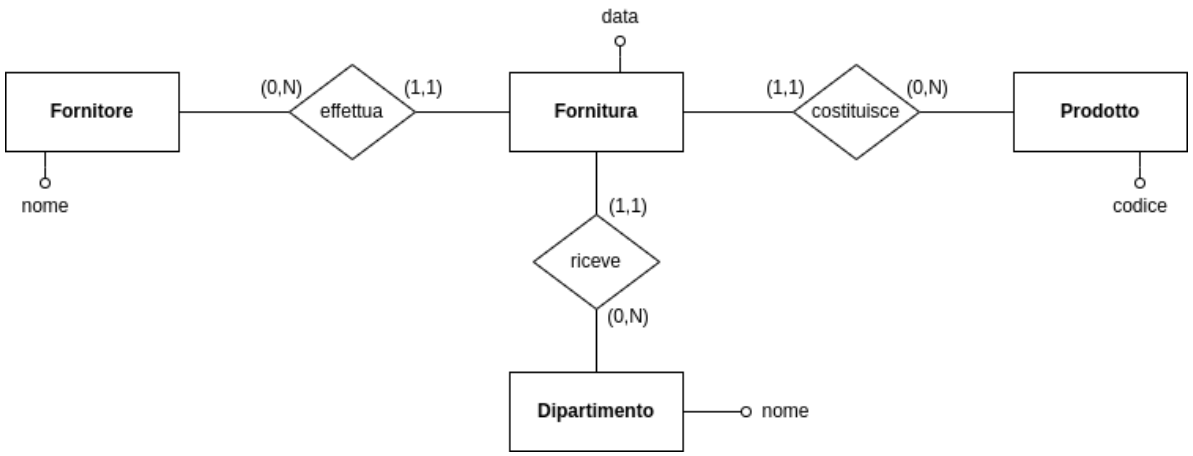
- Tuttavia, tale soluzione risulta essere **errata**: le triple di tale relationship saranno comunque nella forma

  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 effettuate, 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.

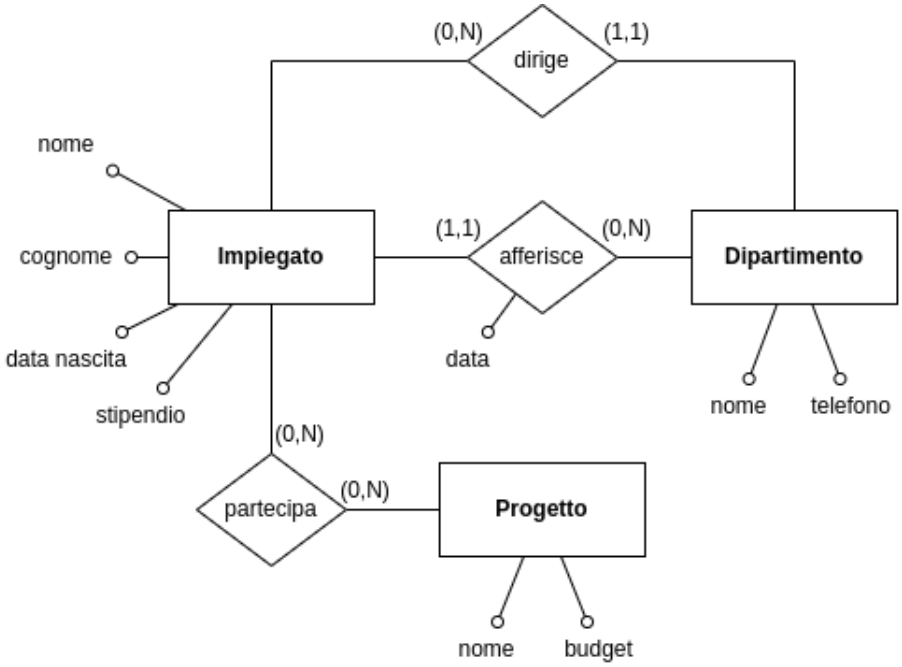
     Di ogni impiegato interessa conoscere il nome, il cognome, la data di nascita e lo stipendio attuale, il dipartimento (esattamente uno) al quale afferisce.

     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 Relazioni is-a e Generalizzazioni tra entity

Definition 11. Relazione is-a tra entity

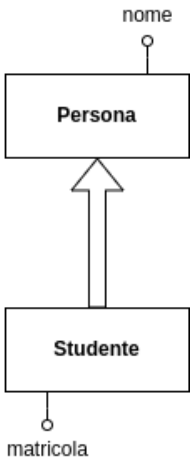
Definiamo come **relazione is-a tra due entity** E **ed** F una relazione indicante che F sia un'entity derivata da E, **ereditandone** tutte le istanze, estendendole con aggiuntivi attributi e relationship.

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**

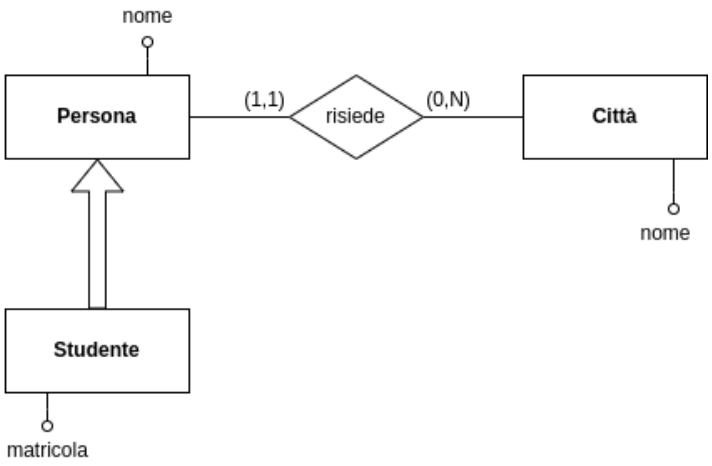
Esempio:

- Consideriamo il seguente diagramma ER:

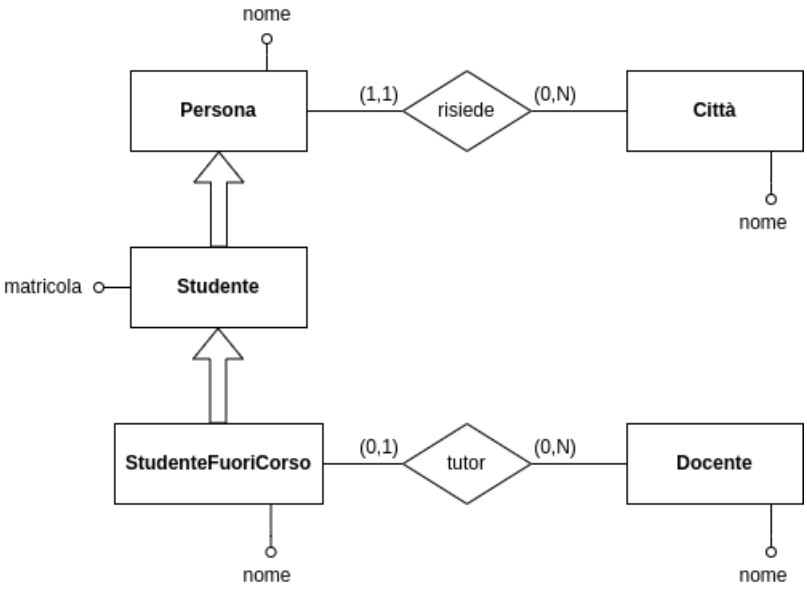


  In tal caso, ogni istanza di Studente è anche un'istanza di Persona, dunque ogni studente possiede anche un nome, oltre alla matricola aggiuntiva. Ovviamente, il viceversa non vale, dunque una persona non è detto che sia uno studente.

- 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

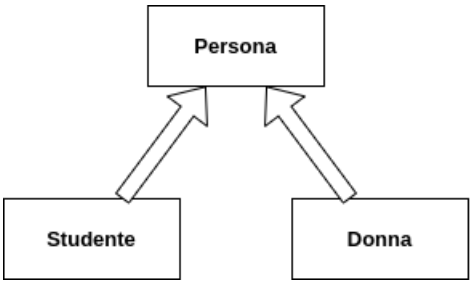


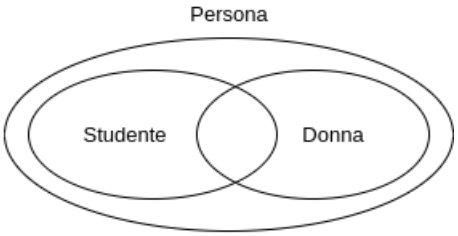
Observation 5. Derivazione multipla

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

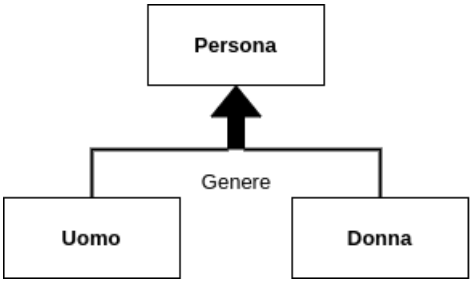
Definition 12. Generalizzazione completa e non completa

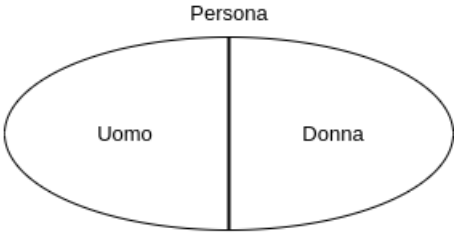
Definiamo come **generalizzazione** un particolare tipo di **relazione is-a** dove le entity derivate dalla base sono **disgiunte** tra loro.

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**.

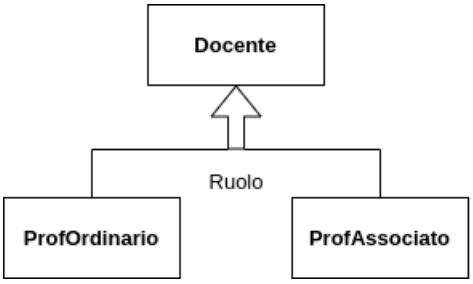
Esempi:

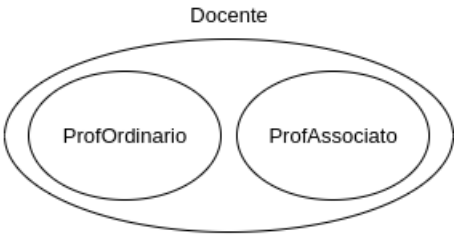
- Nel seguente diagramma ER, non può esistere un'istanza di Persona che non sia un'istanza di Uomo o un'istanza di Donna (generalizzazione completa)



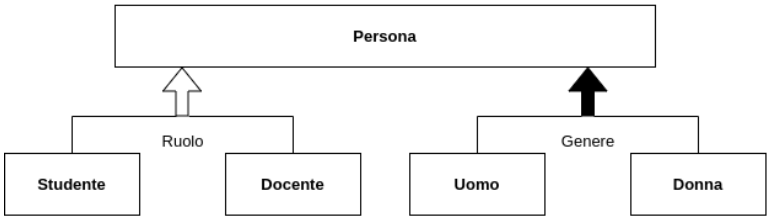


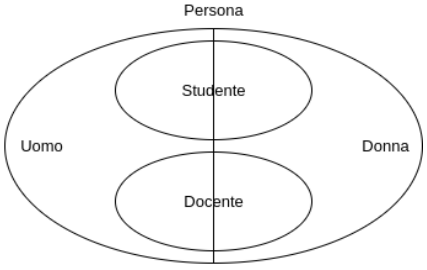
- Nel seguente diagramma ER, può esistere un'istanza di Docente che non sia un'istanza di ProfOrdinario o un'istanza di ProfAssociato (generalizzazione non completa)





- 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 Relazioni is-a 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 le istanze, estendendole con aggiuntivi attributi e relationship.

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

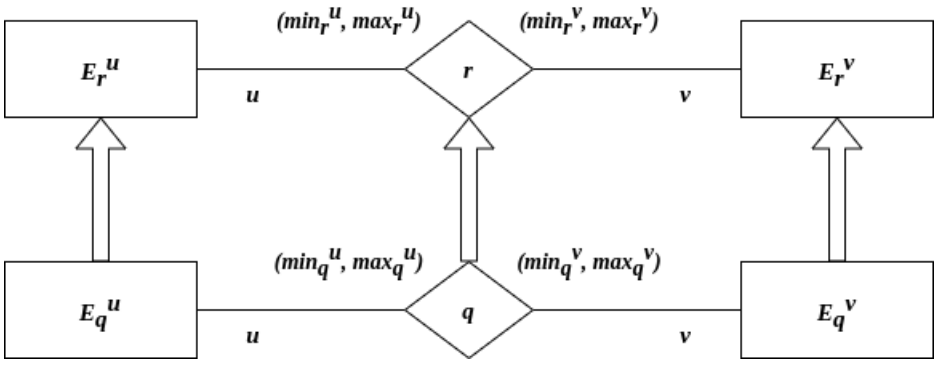
Inoltre, definiamo r come **relationship base** e q come **relationship derivata**

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.

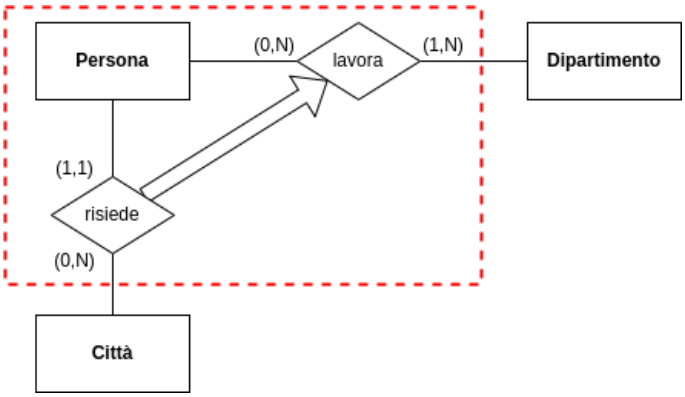
Data una **relationship** q **derivata da** r, affinché tale relazione is-a **possa esistere**, è necessario che:

- 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:
  - $max_q^v \leq max_r^v$
  - $E_q^v = E_r^v \implies min_q^v \leq min_r^v$

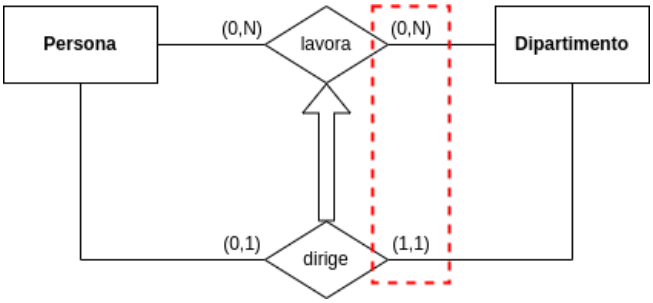


Esempi:

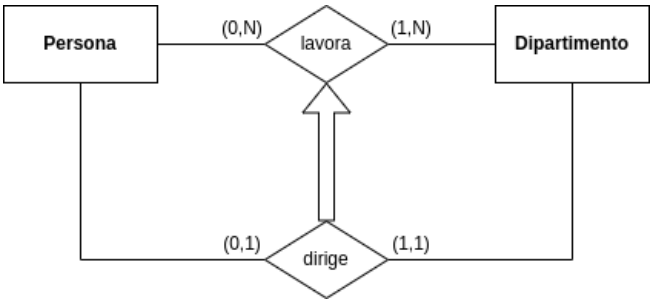
1.
   - Consideriamo il seguente diagramma ER. Notiamo facilmente come la realtà modellata violi le condizioni di esistenza della relazione is-a tra le relationship "risiede" e "lavora", poiché tali relationship sono chiaramente non dello stesso tipo, modellando una realtà non avente alcun senso logico.



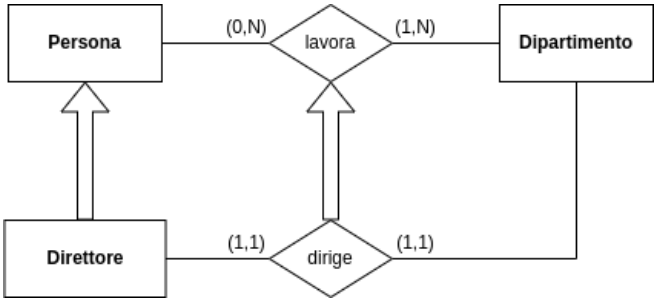
2.
   - Consideriamo il seguente diagramma ER. La realtà modellata è la seguente:
     - Ogni Persona può lavorare in nessuno o più dipartimenti
     - In ogni Dipartimento possono lavorare nessuna o più persone
     - Ogni Persona può dirigere nessuno o il dipartimento in cui lavora
     - Ogni Dipartimento 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.
   - Restringendo il vincolo di molteplicità del ruolo dipartimento nella relationship "lavora", tale incongruenza logica viene risolta



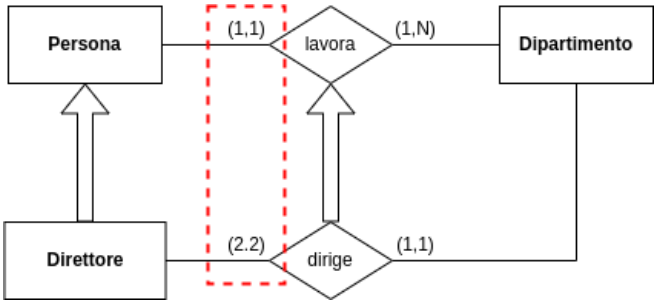
3. • Consideriamo il seguente diagramma ER. La realtà modellata è la seguente:
   - Ogni Persona può lavorare in nessuno o più dipartimenti
   - In ogni Dipartimento possono lavorare una o più Persone
   - Ogni Direttore dirige il Dipartimento in cui lavora
   - Ogni Dipartimento è diretto da un Direttore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

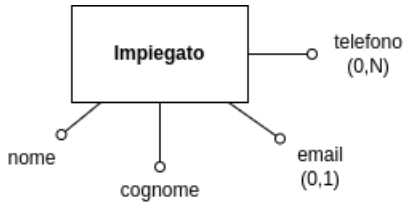
Definiamo come **vincolo di cardinalità su un attributo** un vincolo di integrità che esprime il **numero di valori** assunti dall'attributo designato di ogni istanza di un'entity.

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 possedere nessuna o una sola email, nessun o più numeri di telefono e deve possedere 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

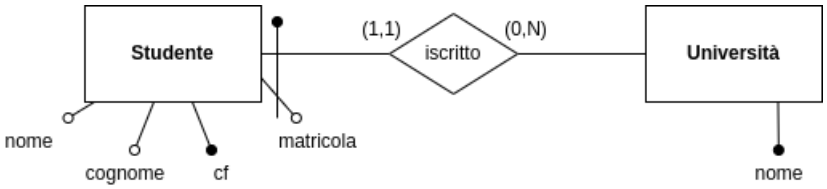
Gli identificatori utilizzati per un vincolo di identificazione devono essere **insiemi minimali di attributi**, ossia non deve poter esistere un sottoinsieme di attributi di tale identificatore che possa fungere a sua volta come identificatore

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.

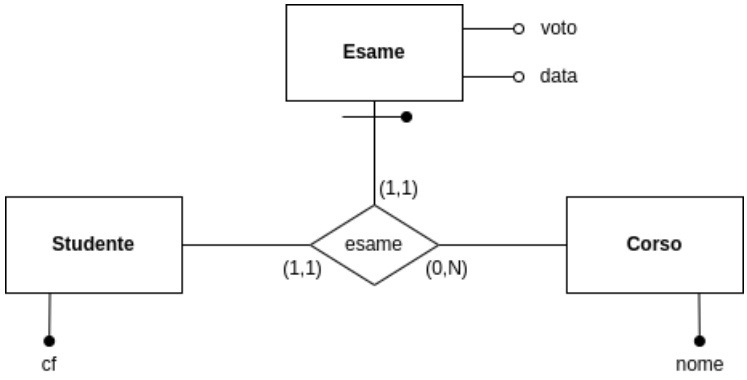
     In particolare, vogliamo che non esistano due studenti con lo stesso codice fiscale, non esistano due università con lo stesso nome e che non esistano due studenti aventi la stessa matricola frequentanti la stessa università

- 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:

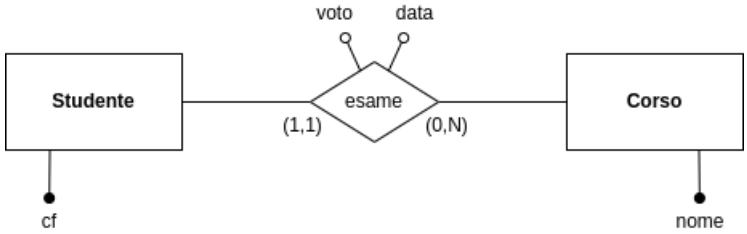


2. - Si vogliono rappresentare degli studenti (con relativo nome), dei corsi (con relativo nome) e gli esami sostenuti dagli studenti (con relativo voto e data)

     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 estensionale, 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)



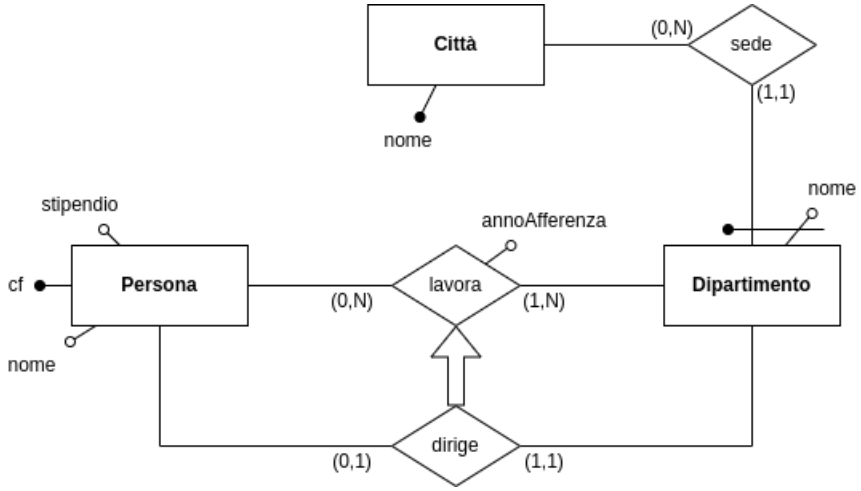
Definition 16. Vincolo esterno

Definiamo come **vincolo esterno** un vincolo di integrità inserito all'interno del dizionario dei dati e non rappresentabile attraverso il diagramma ER, imponendo ulteriori restrizioni ai livelli estensionali ammessi.

Ogni vincolo esterno è definito tramite un **identificatore univoco**, ossia un nome che lo rappresenti, ed un'**asserzione**, ossia la condizione imposta

Esempio:

- Consideriamo il seguente diagramma ER



- Alcuni vincoli esterni imponibili sono:
  - **V.dirige.afferenza**: Per ogni istanza `(p: Persona, d: Dipartimento)` della relationship "dirige", l'istanza `(p: Persona, d: Dipartimento)` deve avere un valore v per l'attributo annoAfferenza tale che $v \leq$ annoCorrente -5, dove annoCorrente denota l'istanza di tipo intero rappresentante 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*, dati:
    - Il valore $stip_{dir}$ dell'attributo stipendio di *dir*
    - Il valore $stip_p$ dell'attributo stipendio di p

    si deve avere $stip_{dir} \geq stip_p$.

Observation 8

Attualmente, i vincoli esterni risultano complessi da esprimere per via dell'utilizzo del linguaggio naturale.

In seguito, verrà utilizzata la **logica del primo ordine** per esprimere tali vincoli, eliminando le ambiguità e alleggerendo la sintassi.

2.2 Diagramma UML degli use-case

Definition 17. 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 accessibili da un gruppo omogeneo di **utenti** del sistema.

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

Definition 18. 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 19. 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:

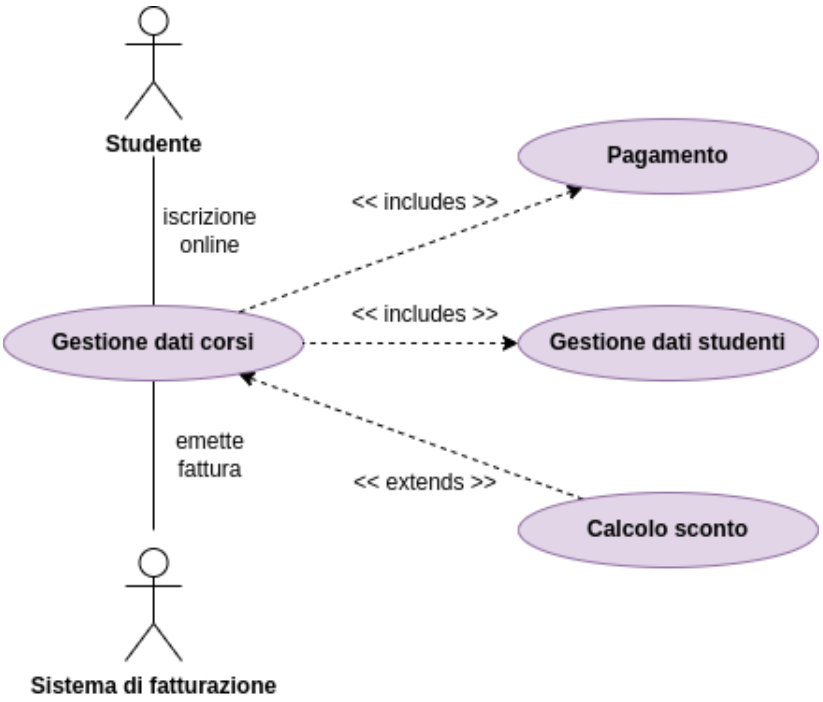
Definition 20. Inclusione ed Estensione

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

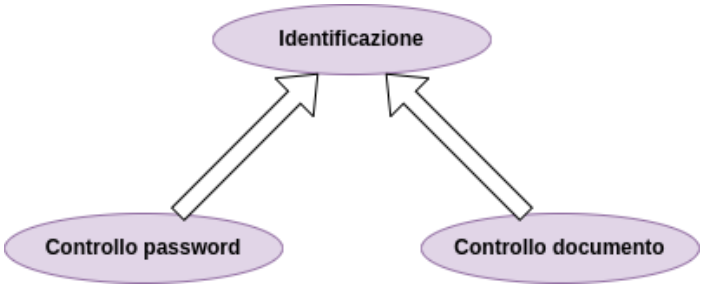


Definition 21. Generalizzazione tra use-case

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 funzionalità ed estendendole 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

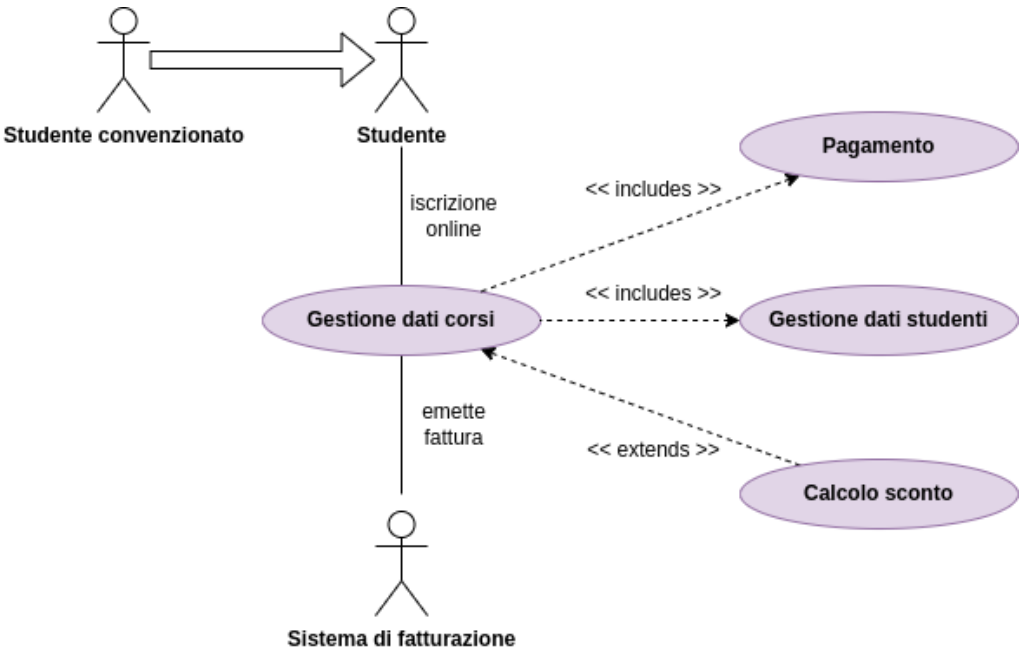


Definition 22. Generalizzazione tra attori

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, estendendoli con aggiuntivi.

Esempio:

- Riprendendo l'esempio precedente, uno Studente Convenzionato è un caso particolare dell'attore Studente, poiché esso è coinvolto negli stessi use-case di quest'ultimo



2.2.1 Specifiche degli use-case

Definition 23. Spefiche di uno use-case

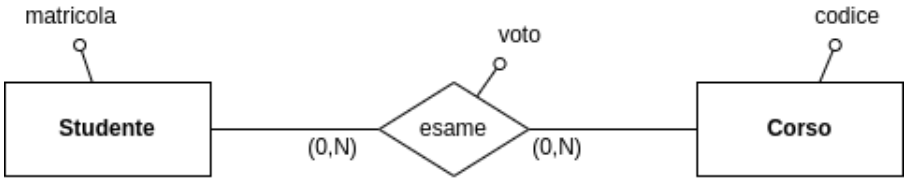
Dato uno use case, definiamo come **specifica di uno use-case** un insieme di descrizioni di tutte le operazioni coinvolte in tale use-case.

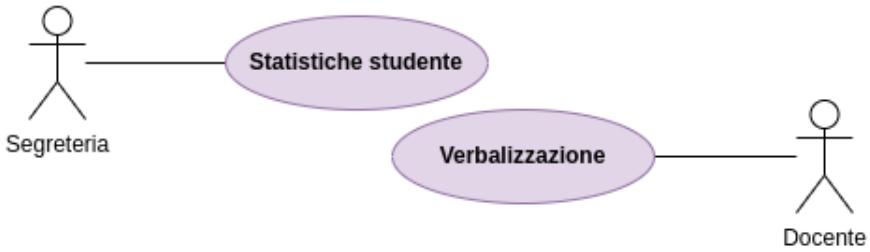
Ogni descrizione di una operazione è composta da:

- Una **segnatura**, composta dal **nome** dell'operazione, un insieme di **argomenti**, il **dominio** di essi e del **valore di ritorno** dall'operazione (se esistente)
- Una serie di **pre-condizioni**, ossia le condizioni sugli argomenti e sul livello estensionale del sistema che devono valere all'avvio dell'esecuzione dell'operazione, affinché il suo comportamento sia definito
- Una serie di **post-condizioni**, ossia le condizioni sul livello estensionale del sistema che devono valere al termine dell'esecuzione dell'operazione e definizione dell'eventuale valore di ritorno

Esempio:

- Consideriamo il seguente diagramma ER e il seguente diagramma degli use-case





- Definiamo quindi le seguenti specifiche degli use-case:

  Statistiche studente:

  ```
  mediaVoti(s: Studente): Reale [18, 31]
      Pre-condizione: l'istanza s è coinvolta in almeno un'istanza della
      relationship esame

      Post-condizione: 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
      Pre-condizione: il livello estensionale dei dati definisce almeno
      una istanza di entity Studente

      Post-condizione: result è pari al numero di istanze di relationship
      esame definite nel livello estensionale diviso per il numero di
      istanze di entity Studente
  end
  ```

  Verbalizzazione:

  ```
  verbalizzaEsame(s: Studente, c: Corso, v: [18, 31]):
      Pre-condizione: l'istanza s non è coinvolta in alcuna istanza della
      relationship esame con l'istanza c

      Post-condizione: viene creata l'istanza (s: Studente, c: Corso)
      della relationship esame con valore v per l'attributo voto
  end
  ```

Capitolo 3

Logica del primo ordine (FOL)

Definition 24. Logica del primo ordine (FOL)

La **logica del primo ordine (first order logic - FOL)** è un linguaggio formale utilizzato per rappresentare informazioni e derivare conseguenze da esse.

In particolare, la FOL è definita da:

- **Sintassi**, ossia l'insieme delle sequenze finite di simboli ammesse dal linguaggio, dette **formule**, definendo la struttura di esse, dove ogni simbolo appartiene ad un insieme prefissato, detto **alfabeto**
- **Semantica**, ossia il significato assunto da ogni formula, in particolare la sua verità nei diversi mondi possibili

3.1 Sintassi della FOL

Proposition 4. Alfabeto della FOL

L'alfabeto utilizzato nella logica del primo ordine è costituito da:

- Un insieme $\mathcal{V}$ di **variabili**
- Un insieme $\mathcal{F}$ di **simboli di funzione**, ognuno dei quali ha associata la sua **arità**, ossia il numero di argomenti
- Un insieme $\mathcal{P}$ di **simboli di predicato** (o **relazioni**), ognuno dei quali ha associata la sua **arità**, ossia il numero di argomenti.
- I **connettivi logici** $\neg, \vee, \wedge, \rightarrow, \leftrightarrow$
- Il **quantificatore universale** $\forall$ e il **quantificatore esistenziale** $\exists$
- I simboli speciali di **parentesi**, ossia "(" e ")", e di **virgola**, ossia ","

Observation 9

Per convenzione, assumeremo che:

- Il **simbolo di funzione** f e la sua arità k viene rappresentato in $\mathcal{F}$ con la notazione f/k
- Il **simbolo di predicato** p e la sua arità h viene rappresentato in $\mathcal{P}$ con la notazione p/h
- I simboli di funzione di arità 0 vengono detti anche **simboli di costante**
- Il predicato $=/2$ sia **sempre presente** in $\mathcal{P}$

Esempi:

- Un esempio di insieme di simboli di funzione corrisponde a

$$\mathcal{F} = \{\text{zero}/0, \text{succ}/1, \text{socrate}/0, \text{padre}/1\}$$

  dove:

  - zero/0 è un simbolo di costante rappresentante il numero naturale 0
  - succ(X) rappresenta il numero naturale $X + 1$
  - socrate/0 è un simbolo di costante rappresentante l'individuo Socrate
  - padre(X) rappresenta il padre dell'individuo X

- Un esempio di insieme di simboli di predicato corrisponde a:

$$\mathcal{P} = \{\text{doppio}/2, \text{somma}/3, \text{uomo}/1, \text{mortale}/1\}$$

  dove:

  - doppio(X, Y) indica che il numero naturale Y è il doppio del numero naturale X
  - somma(X, Y, Z) indica che $X + Y = Z$
  - uomo(X) indica che l'individuo X è un uomo
  - mortale(X) indica che l'individuo X è un mortale

Definition 25. Linguaggio dei termini

L'**insieme dei termini** $\mathcal{T}$ è definito induttivamente come:

- Ogni **variabile** in $\mathcal{V}$ è un termine
- Ogni **simbolo di costante** in $\mathcal{F}$ è un termine
- Se $f/k \in \mathcal{F}$, dove $k > 0$, e $t_1, \ldots, t_k \in \mathcal{T}$, allora $f(t_1, \ldots, t_k) \in \mathcal{T}$

Attenzione: ogni termine denota un *oggetto di interesse* del mondo che si sta modellando. Tuttavia, *quale* oggetto di interesse sia denotato non è stabilito dalla sintassi, bensì dalla semantica.

Esempi:

- Dato $\mathcal{V} = \{X, \text{MiaVariabile}\}$ e dato $\mathcal{F} = \{\text{zero}/0, \text{succ}/1, \text{socrate}/0, \text{padre}/1\}$, le seguenti sequenze di simboli sono termini:
  - zero
  - MiaVariabile
  - succ(zero)
  - padre(padre(socrate))
  - padre(succ(X))
  - succ(succ(zero))

Definition 26. Linguaggio delle formule

Una **formula** è definita induttivamente come:

- Se $p/k \in \mathcal{P}$ e $t_1, \ldots, t_k \in \mathcal{T}$, allora $p(t_1, \ldots, t_k)$ è una formula
- Se ϕ, ψ sono formule, allora
  - (ϕ) e $\neg\phi$
  - $\phi \vee \psi$ e $\phi \wedge \psi$
  - $\phi \to \psi$ e $\phi \leftrightarrow \psi$

  sono formule
- Se ϕ è una formula e $X \in \mathcal{V}$, allora $\forall X\phi$ e $\exists X\phi$ sono formule

Observation 10

Per convenzione, assumeremo che $X = Y$ rappresenti la formula $= (X, Y)$ e che $X \neq Y$ rappresenti la formula $\neg(= (X, Y))$

Observation 11

Alcune formule possono essere ben definite ma avere un significato logico completamente errato, il quale dipende dalla **semantica** della formula, non dalla sua sintassi

Esempi:

- Dati:
  - $\mathcal{F} = \{\text{zero}/0, \text{succ}/1, \text{socrate}/0, \text{padre}/1\}$
  - $\mathcal{P} = \{\text{doppio}/2, \text{somma}/3, \text{uomo}/1, \text{mortale}/1\}$

  le seguenti sequenze di simboli sono formule:
  - doppio(succ(succ(zero)), X)

- $\forall X$ doppio(succ(succ(zero)), X)
- somma(succ(zero), zero, succ(zero))
- $\forall X \forall Y$ somma(X, X, Y) $\rightarrow$ doppio(X, Y)
- $(\forall X \exists Y$ doppio(X,Y)) $\wedge$ $(\forall I \forall J \exists K$ somma(I, J, K))
- mortale(socrate) $\wedge$ mortale(padre(socrate))
- $(\forall X$ uomo(X) $\rightarrow$ mortale(X)) $\wedge$ uomo(socrate)
- $\forall X$ uomo(X) $\rightarrow$ uomo(padre(X))
- $\forall X$ uomo(socrate)
- X = socrate

3.2 Semantica della FOL

Proposition 5. Semantica della logica proposizionale

Data una formula φ, nella **logica proposizionale** (dunque non del primo ordine), definiamo come:

- **Lettere proposizionali** l'insieme di **formule atomiche** che compongono φ, ossia le sottoformule prive di connettivi logici
- **Interpretazione** una funzione I che assegna un **valore di verità** ad ogni lettera proposizionale di φ
- **Funzione di valutazione** una funzione che calcola il valore di verità della formula φ ricorsivamente tramite il valore di verità delle sottoformule $\varphi_1, \ldots, \varphi_n$ rispetto ad un'interpretazione I.

  Se la formula φ è **vera** nell'interpretazione I, allora I è un **modello** di φ, indicato come $I \models \varphi$.

Definition 27. Soddisfacibilità, validità e insoddisfacibilità

Data una formula φ, nella **logica proposizionale** (dunque non del primo ordine), tale formula viene detta:

- **Soddisfacibile** se esiste un'interpretazione I tale che $I \models \varphi$
- **Valida** se per ogni un'interpretazione I tale che $I \models \varphi$
- **Insoddisfacibile** non esiste un'interpretazione I tale che $I \models \varphi$

Esempio:

- Consideriamo la seguente formula $\varphi : a \wedge (b \vee c)$, dove lettere proposizionali di φ corrispondono a $\{a, b, c\}$

- Data l'interpretazione $I : I(a) = true, I(b) = true, I(c) = false$, si ha che:
  - La formula atomica b è vera
  - La formula atomica c è vera
  - La sottoformula $(b \vee c)$ è vera, per semantica del connettivo $\vee$
  - La formula atomica a è vera
  - La formula $a \wedge (b \vee c)$ è vera, per semantica del connettivo $\wedge$

  dunque, ne segue che $I \models \varphi$, implicando che φ sia soddisfacibile

Proposition 6. Semantica nella FOL

Data una formula φ, nella **logica del primo ordine** viene:

1. Definita la nozione di **interpretazione** valutando le formule atomiche che costituiscono φ
2. Definita la modalità di valutazione di φ **in base** ad una determinata interpretazione I
3. Si stabilisce il significato di ogni formula **senza riferimento** ad una determinata interpretazione

A tal scopo, vengono evidenziati due nozioni di valutazione:

- **Valutazione dei termini**, ossia la valutazione dei termini atomici, composta da una **pre-interpretazione** ed un **assegnamento delle variabili**, e la valutazione dei termini complessi
- **Valutazione delle formule**, ossia la valutazione delle formule atomiche, in base ad un'interpretazione, e la valutazione della formula nel suo totale

Definition 28. Pre-interpretazione

Sia $\mathcal{F}$ un insieme di simboli di funzione. Una **pre-interpretazione** `preI` per $\mathcal{F}$ è costituita da:

- Un insieme $\mathcal{D}$ non vuoto finito o non finito detto **dominio di interpretazione**
- Una corrispondenza che associa ad ogni $f/k \in \mathcal{F}$ una **funzione totale**, intesa nel senso matematico, del tipo $\mathcal{D}^k \to \mathcal{D}$, indicato come `preI(`$f$`)`.

  In particolare, se $k = 0$, dunque se f è un simbolo di costante, tale funzione associa a $f/0$ un elemento di $\mathcal{D}$

Esempio:

- Dato $\mathcal{F} = \{\text{zero}/0, \text{succ}/1\}$, definiamo la pre-interpretazione `preNAT` per $\mathcal{F}$ come:
  - $\mathcal{D} = \mathbb{N}$
  - `preNAT`$(\text{zero}) = 0 \in \mathcal{D}$

  - $\texttt{preNAT}(\text{succ})$ è la funzione $\mathcal{D}^1 \to \mathcal{D}$ definita come:
    * $\texttt{preNAT}(\text{succ})(0) = 1$
    * $\texttt{preNAT}(\text{succ})(1) = 2$
    * $\texttt{preNAT}(\text{succ})(2) = 3$
    * ...

Definition 29. Assegnamento delle variabili

Sia $\mathcal{V}$ un insieme di variabili e sia $\texttt{preI}$ una pre-interpretazione con dominio $\mathcal{D}$. Un **assegnamento delle variabili** $\mathcal{V}$ per $\texttt{preI}$ è una funzione $\mathcal{V} \to \mathcal{D}$ che assegna ad ogni variabile in $\mathcal{V}$ un elemento di $\mathcal{D}$

Esempio:

- Dato $\mathcal{V} = \{X, Y, Z\}$ e data la pre-interpretazione $\texttt{preNAT}$ vista nell'esempio precedente dove $\mathcal{D} = \mathbb{N}$, la funzione $W : \mathcal{V} \to \mathcal{D}$ tale che
  - $W(X) = 3$
  - $W(Y) = 6$
  - $W(Z) = 4$

  è un assegnamento delle variabili $\mathcal{V}$ per $\texttt{preNAT}$

Proposition 7. Valutazione dei termini

Dati gli insiemi $\mathcal{V}, \mathcal{F}$, sia $\mathcal{T}$ l'insieme di **tutti i termini generabili** tramite $\mathcal{V}$ e $\mathcal{F}$.

Data una **pre-interpretazione** $\texttt{preI}$ su un dominio $\mathcal{D}$ e un **assegnamento di variabili** S per $\texttt{preI}$, la funzione

$$\texttt{pre-eval}^{\texttt{preI},S} : \mathcal{T} \to \mathcal{D}$$

è definita induttivamente come:

- **Caso base (termini atomici)**:
  - Se $X \in \mathcal{V}$, allora $\texttt{pre-eval}^{\texttt{preI},S}(X) = S(X)$
  - Se $c/0 \in \mathcal{F}$, allora $\texttt{pre-eval}^{\texttt{preI},S}(c) = \texttt{preNAT}(c)$
- **Caso induttivo (termini complessi)**:
  - Se $f/k \in \mathcal{F}$, dove $k > 0$, e $t_1, \ldots, t_k$ sono termini, allora

$$\texttt{pre-eval}^{\texttt{preI},S}(f(t_1, \ldots, t_k)) = \texttt{preI}(f)(\texttt{pre-eval}^{\texttt{preI},S}(t_1), \ldots, \texttt{pre-eval}^{\texttt{preI},S}(t_k))$$

Esempio:

- Riprendiamo la pre-interpretazione $\texttt{preNAT}$ e l'assegnamento W dell'esempio precedente

- L'insieme $\mathcal{T}$ dei termini generabili da $\mathcal{V}$ e $\mathcal{F}$ corrisponde a:

$$\mathcal{T} = \{\text{zero}, X, Y, Z, \text{succ}(\text{zero}), \text{succ}(X), \text{succ}(Y), \text{succ}(Z), \text{succ}(\text{succ}(zero)),$$
$$\text{succ}(\text{succ}(X)), \text{succ}(\text{succ}(Y)), \text{succ}(\text{succ}(Z)), \text{succ}(\text{succ}(\text{succ}(zero))),$$
$$\text{succ}(\text{succ}(\text{succ}(X))), \text{succ}(\text{succ}(\text{succ}(Y))), \text{succ}(\text{succ}(\text{succ}(Z))), \ldots\}$$

- Poiché $zero/0 \in \mathcal{F}$, ne segue che:

$$\texttt{pre-eval}^{\texttt{preNAT},W}(zero) = \texttt{preNAT}(zero) = 0$$

- Poiché $succ/1 \in \mathcal{F}$ e poiché $W(X) = 3$, ne segue che:

$$\texttt{pre-eval}^{\texttt{preNAT},W}(succ(succ(X))) = \texttt{preNAT}(succ)(\texttt{pre-eval}^{\texttt{preNAT},W}(succ(X))) =$$
$$= \texttt{preNAT}(succ)(\texttt{preNAT}(succ)(\texttt{pre-eval}^{\texttt{preNAT},W}(X))) =$$
$$= \texttt{preNAT}(succ)(\texttt{preNAT}(succ)(W(X))) =$$
$$= \texttt{preNAT}(succ)(\texttt{preNAT}(succ)(3)) = \texttt{preNAT}(succ)(4) = 5$$

Definition 30. Interpretazione nella FOL

Sia $\mathcal{F}$ un insieme di simboli di funzione. Un'**interpretazione** `I` è costituita da:

- Una **pre-interpretazione** `preI`, la quale a sua volta definisce un dominio $\mathcal{D}$ ed una funzione su $\mathcal{D}$ per ogni simbolo di funzione in $\mathcal{F}$
- Una corrispondenza che associa ad ogni simbolo di predicato $p/k \in \mathcal{P}$ una **relazione** $I(p) \subseteq \mathcal{D}^k$.

  In particolare, poiché il simbolo di predicato $=$ è sempre presente in $\mathcal{P}$, tale corrispondenza deve assegnare a $=$ la relazione

$$\{(d, d) \mid d \in \mathcal{D}\} \subseteq D^2$$

Observation 12

Una pre-interpretazione ed un'interpretazione non includono un assegnamento di variabili, il quale andrà fatto esternamente

Esempio:

- Riprendendo la pre-interpretazione `preNAT` con gli insiemi $\mathcal{F} = \{\text{zero}/0, \text{succ}/1\}$ e $\mathcal{P} = \{\text{doppio}/2, \text{somma}/3\}$, definiamo l'interpretazione `NAT` come:
  - La sua pre-interpretazione è `preNAT`
  - `NAT`(doppio)$= \{(0,0), (1,2), (2,4), \ldots\} = \{(x, y) \mid y = 2x\}$
  - `NAT`(somma)$= \{(0,0,0), (0,1,1), (1,0,1), \ldots\} = \{(x, y, z) \mid x + y = z\}$

Proposition 8. Valutazione delle formule

Dati gli insiemi $\mathcal{V}, \mathcal{F}, \mathcal{P}$, sia Φ l'insieme di **tutte le formule generabili** da $\mathcal{V}, \mathcal{F}$ e $\mathcal{P}$.

Data un'interpretazione $\mathtt{I}$ definita su $\mathcal{P}$ che includa una pre-interpretazione $\mathtt{preI}$ definita su $\mathcal{F}$ e dato un assegnamento S delle variabili in $\mathcal{V}$ per $\mathtt{preI}$, la funzione

$$\mathtt{eval}^{\mathtt{preI},S} : \Phi \to \{true, false\}$$

è definita induttivamente come segue:

- **Caso base (formule atomiche)**:
  - Dato $p/k \in \mathcal{P}$, se $t_1, \ldots, t_k$ sono termini allora

    $$\mathtt{eval}^{\mathtt{I},S}(p(t_1, \ldots, t_k)) = \mathtt{I}(p)(\mathtt{pre\text{-}eval}^{\mathtt{preI},S}(t_1), \ldots, \mathtt{pre\text{-}eval}^{\mathtt{preI},S}(t_k))$$

- **Caso induttivo (formule complesse)**:
  - Se ϕ e ψ sono formule allora
    - $\mathtt{eval}^{\mathtt{I},S}(\neg\phi) = \begin{cases} true & \text{se } \mathtt{eval}^{\mathtt{I},S}(\phi) = false \\ false & \text{altrimenti} \end{cases}$
    - $\mathtt{eval}^{\mathtt{I},S}(\phi \vee \psi) = \begin{cases} true & \text{se } \mathtt{eval}^{\mathtt{I},S}(\phi) = true \vee \mathtt{eval}^{\mathtt{I},S}(\psi) = true \\ false & \text{altrimenti} \end{cases}$
    - $\mathtt{eval}^{\mathtt{I},S}(\phi \wedge \psi) = \begin{cases} true & \text{se } \mathtt{eval}^{\mathtt{I},S}(\phi) = true \wedge \mathtt{eval}^{\mathtt{I},S}(\psi) = true \\ false & \text{altrimenti} \end{cases}$
    - $\mathtt{eval}^{\mathtt{I},S}(\phi \to \psi) = \mathtt{eval}^{\mathtt{I},S}(\neg\phi \vee \psi)$
    - $\mathtt{eval}^{\mathtt{I},S}(\phi \leftrightarrow \psi) = \mathtt{eval}^{\mathtt{I},S}((\phi \to \psi) \wedge (\psi \to \phi))$
  - Dato $V \in \mathcal{V}$, se ϕ è una formula allora
    - $\mathtt{eval}^{\mathtt{I},S}(\exists V\phi) = \begin{cases} true & \text{se } \exists d \in \mathcal{D} \mid \mathtt{eval}^{\mathtt{I},S[V/d]}(\phi) = true \\ false & \text{altrimenti} \end{cases}$
    - $\mathtt{eval}^{\mathtt{I},S}(\forall V\phi) = \begin{cases} true & \text{se } \forall d \in \mathcal{D}, \mathtt{eval}^{\mathtt{I},S[V/d]}(\phi) = true \\ false & \text{altrimenti} \end{cases}$

    dove $S[V/d]$ indica un assegnamento uguale ad S tranne per il fatto che alla variabile V venga assegnato il valore d

Esempi:

- Riprendendo l'interpretazione $\mathtt{NAT}$ con gli insiemi $\mathcal{F} = \{\text{zero}/0, \text{succ}/1\}$ e $\mathcal{P} = \{\text{doppio}/2, \text{somma}/3\}$, consideriamo il seguente assegnamento W sulle variabili $\mathcal{V} = \{X, Y, I, J, K\}$
  - $W(X) = 3, W(Y) = 6, W(I) = W(J) = W(K) = 4$

- Valutiamo la seguente formula:

$$\mathsf{eval}^{\mathsf{NAT},W}(doppio(succ(succ(zero)),X)) =$$
$$= \mathsf{NAT}(doppio)(\mathsf{pre\text{-}eval}^{\mathsf{preNAT},W}(succ(succ(zero))),\mathsf{pre\text{-}eval}^{\mathsf{preNAT},W}(X)) =$$
$$= \mathsf{NAT}(doppio)(\mathsf{preNAT}(succ)(\mathsf{pre\text{-}eval}^{\mathsf{preNAT},W}(succ(zero))),W(X)) =$$
$$= \mathsf{NAT}(doppio)(\mathsf{preNAT}(succ)(\mathsf{preNAT}(succ)(\mathsf{pre\text{-}eval}^{\mathsf{preNAT},W}(zero))),3) =$$
$$= \mathsf{NAT}(doppio)(\mathsf{preNAT}(succ)(\mathsf{preNAT}(succ)(\mathsf{preNAT}(zero))),3) =$$
$$= \mathsf{NAT}(doppio)(\mathsf{preNAT}(succ)(\mathsf{preNAT}(succ)(0)),3) =$$
$$= \mathsf{NAT}(doppio)(\mathsf{preNAT}(succ)(1),3) =$$
$$= \mathsf{NAT}(doppio)(2,3) =$$
$$= false$$

Definition 31. Formula chiusa ed aperta

Una formula ϕ viene detta **chiusa** se tutte le variabili al suo interno sono **quantificate** da un quantificatore $\exists$ o un quantificatore $\forall$. Se tale condizione non è verificata, la formula ϕ viene detta **aperta**.

Observation 13

Durante la valutazione di una formula chiusa ϕ, l'assegnamento delle variabili S associato a tale valutazione è **irrilevante**, poiché ogni variabile è quantificata

Esempi:

- Riprendendo l'esempio precedente, consideriamo la seguente formula:

$$\mathsf{eval}^{\mathsf{NAT},W}(\exists X\ doppio(succ(succ(zero)),X))$$

- Nonostante si abbia $W(X) = 3$, la quantificazione $\exists$ sulla variabile X "sovrascrive" l'assegnamento W.
- Di conseguenza, si ha che

$$\mathsf{eval}^{\mathsf{NAT},W}(\exists X\ doppio(succ(succ(zero)),X)) = true$$

  poiché $\exists d := 4 \in \mathcal{D} \mid \mathsf{eval}^{\mathsf{NAT},W[X/1]}(doppio(succ(succ(zero)),X)) = true$
- Analogamente, si avrà che:

$$\mathsf{eval}^{\mathsf{NAT},W}(\forall X\ doppio(succ(succ(zero)),X)) = false$$

  poiché $\exists d := 1 \in \mathcal{D} \mid \mathsf{eval}^{\mathsf{NAT},W[X/1]}(doppio(succ(succ(zero)),X)) = false$

Definition 32. Soddisfacibilità, validità e insoddisfacibilità nella FOL

Data una formula ϕ, nella FOL tale formula viene detta:

- **Soddisfacibile** se esistono un'interpretazione $\mathtt{I}$ e un assegnamento di variabili S tale che $\mathtt{eval}^{\mathtt{I},S}(\phi) = true$
- **Valida** se per ogni interpretazione $\mathtt{I}$ e ogni assegnamento di variabili S si ha che $\mathtt{eval}^{\mathtt{I},S}(\phi) = true$
- **Insoddisfacibile** se per ogni interpretazione $\mathtt{I}$ e ogni assegnamento di variabili S si ha che $\mathtt{eval}^{\mathtt{I},S}(\phi) = false$

Observation 14

Per convenzione, nella FOL vengono utilizzate le seguenti **regole di precedenza** per la valutazione dei connettivi e dei quantificatori:

1. $\neg$
2. $\wedge, \vee$
3. $\forall, \exists$
4. $\rightarrow, \leftrightarrow$

Esempio:

- La formula
$$\forall X\, P(X) \vee S(X) \rightarrow \exists Y\, Q(X,Y) \wedge \neg R(Y)$$
viene valutata come
$$\forall X\, ((P(X) \vee S(X)) \rightarrow (\exists Y\, (Q(X,Y) \wedge (\neg(R(Y))))))$$

3.3 FOL nell'analisi concettuale

3.3.1 Alfabeto di ER in FOL

Observation 15

Un **diagramma ER** definisce una **formula logica** ϕ corrispondente ad una **congiunzione di blocchi di formule** (ossia varie sottoformule unite da dei connettivi logici $\wedge$), ognuno dei quali definisce la semantica di ogni **modulo** presente nel diagramma (dunque entity, relationship, attributi, domini, relazioni is-a, generalizzazioni, vincoli di identificazione)

In particolare, la formula ϕ definita da un certo diagramma ER sarà definita su un determinato **alfabeto**, ossia su un certo **insieme di simboli di predicato** $\mathcal{P}$ ed un certo **insieme di simboli di funzione** $\mathcal{F}$.

I **simboli di predicato** in $\mathcal{P}$ (assieme alle loro arità) sono **univocamente definiti** dal nome dei diversi moduli e costrutti presenti nel diagramma ER (in particolare dai nomi delle entity, delle relationship, degli attributi e dei domini). Come sempre, assumeremo che $= /2 \in \mathcal{P}$

I **simboli di funzione** in $\mathcal{F}$ (assieme alle loro arità) sono **univocamente definiti** dalle operazioni necessarie per operare sui valori dei domini.

Proposition 9. Simboli di predicato per entity

Dato un diagramma ER, ogni **entity** E presente al suo interno definisce il simbolo di predicato unario $E/1 \in \mathcal{P}$.

In ogni **modello** M della formula ϕ definita dal diagramma ER, le istanze dell'entity E saranno rappresentati dagli elementi e del dominio di interpretazione di M tali che $E(e) = true$

Esempio:

- Supponiamo che un diagramma ER definisca le entity Persona e Azienda
- La formula ϕ definita dal diagramma conterrà occorrenze dei simboli di predicato Persona$/1 \in \mathcal{P}$ e Azienda$/1 \in \mathcal{P}$

$$\mathcal{P} = \{\ldots, \text{Persona}/1, \text{Azienda}/1, \ldots\}$$

- Un modello M di ϕ definirà:
  - L'estensione del simbolo di predicato Persona$/1$

    (es: $M(\text{Persona}) = \{\alpha, \beta, \ldots\}$ rappresenta tutte e sole le istanze dell'entity Persona)
  - L'estensione del simbolo di predicato Azienda$/1$

    (es: $M(\text{Azienda}) = \{\gamma, \delta, \ldots\}$ rappresenta tutte e sole le istanze dell'entity Azienda)

Proposition 10. Simboli di predicato per domini

Dato un diagramma ER, ogni **dominio** *dom* utilizzato al suo interno definisce un simbolo di predicato unario $dom/1 \in \mathcal{P}$.

In ogni **modello** M della formula ϕ definita dal diagramma ER, i valori del dominio *dom* saranno rappresentati dagli elementi d del dominio di interpretazione di M tali che $dom(d) = true$

Esempio:

- Supponiamo che un diagramma ER definisca degli attributi (appartenenti a qualche entity o relationship) aventi come dominio "intero" e "data"
- La formula ϕ definita dal diagramma conterrà occorrenze dei simboli di predicato intero/1$\in \mathcal{P}$ e data/1$\in \mathcal{P}$

$$\mathcal{P} = \{\ldots, \text{intero}/1, \text{data}/1, \ldots\}$$

- Un modello M di ϕ definirà:
  - L'estensione del simbolo di predicato intero/1

    (es: $M(\text{intero}) = \{\ldots, -1, 0, 1, 2, \ldots\}$ rappresenta tutti e soli i valori del dominio intero)
  - L'estensione del simbolo di predicato data/1

    (es: $M(\text{data}) = \{\ldots, 17/03/2022, 08/12/1976, \ldots\}$ rappresenta tutti e soli i valori del dominio data)

Observation 16. Simboli di predicato per domini specializzati

Dato un diagramma ER, ogni **dominio specializzato** *dom_spec* presente al suo interno corrispondente ad una specializzazione di un dominio *dom* definisce il simbolo di predicato unario $dom\_spec/1 \in \mathcal{P}$ e il simbolo di predicato unario $dom/1 \in \mathcal{P}$

In ogni **modello** M della formula ϕ definita dal diagramma ER, i valori dei domini *dom* e *dom_spec* saranno rappresentati dagli elementi d del dominio di interpretazione di M tali che $dom(d) = true, dom\_spec(d) = true$

Esempio:

- Supponiamo che un diagramma ER definisca attributi (appartenenti a qualche entity o relationship) aventi come dominio "intero ≥ 0" e "$[0, 59]$"
- La formula ϕ definita dal diagramma conterrà occorrenze dei simboli di predicato intero/1$\in \mathcal{P}$, intero≥ 0/1$\in \mathcal{P}$ e $[0, 59]/1 \in \mathcal{P}$

$$\mathcal{P} = \{\ldots, \text{intero}/1, \text{intero}\geq 0/1, [0, 59]/1, \ldots\}$$

- Un modello M di ϕ definirà:
  - L'estensione del simbolo di predicato intero/1
  - L'estensione del simbolo di predicato intero≥ 0/1
  - L'estensione del simbolo di predicato $[0, 59]/1$

Observation 17. Simboli di predicato per domini composti

Dato un diagramma ER, ogni **dominio composto** dom presente al suo interno composto dai campi $f_1, \ldots, f_n$ rispettivamente associati ai domini $dom_1, \ldots, dom_n$, definisce:

- Il simbolo di predicato unario $dom/1 \in \mathcal{P}$
- I simboli di predicato unario $dom_1/1, dom_2/1, \ldots, dom_n/1 \in \mathcal{P}$
- I simboli di predicato binario $f_1/2, f_2/2, \ldots, f_n \in \mathcal{P}$

In ogni **modello** M della formula ϕ definita dal diagramma ER si ha che:

- I valori dei domini $dom, dom_1, \ldots, dom_n$ saranno rappresentati dagli elementi $d, d_1, \ldots, d_n$ del dominio di interpretazione di M tali che $dom(d) = true$, $dom_1(d_1) = true, \ldots, dom_n(d_n) = true$
- I valori dei campi $f_1, \ldots, f_n$ saranno rappresentati dagli elementi $d'_1, \ldots, d'_n$ del dominio di interpretazione di M tali che $f_1(d'_1) = true, f_n(d'_n) = true$

Esempio:

- Supponiamo che un diagramma ER definisca attributi (appartenenti a qualche entity o relationship) aventi come dominio "ora", composto dai campi h, m, s rispettivamente associati ai domini $[0, 23], [0.59], [0.59]$
- La formula ϕ definita dal diagramma conterrà occorrenze dei simboli di predicato ora/1, $[0, 23]/2$ e $[0, 59]/2$, $h/1$, $m/1$, $s/1 \in \mathcal{P}$

$$\mathcal{P} = \{\ldots, \text{ora}/1, [0, 23]/2, [0, 59]/2, h/1, m/1, s/1, \ldots\}$$

- Un modello M di ϕ (ad esempio) definirà:
  - $M(\text{ora}) = \{\alpha, \beta, \ldots\}$
  - $M([0, 23]) = \{\gamma, \delta, \ldots\}$
  - $M([0, 59]) = \{\gamma, \delta, \mu, \xi, \ldots\}$
  - $M(h) = \{(\alpha, \gamma), (\beta, \delta), \ldots\}$
  - $M(m) = \{(\alpha, \delta), (\beta, \mu), \ldots\}$
  - $M(s) = \{(\alpha, \xi), (\beta, \delta), \ldots\}$

Proposition 11. Simboli di predicato per attributi di entity

Dato un diagramma ER, ogni **attributo** a **di un'entity** E presente al suo interno definisce il simbolo di predicato binario $a/2 \in \mathcal{P}$.

In ogni **modello** M della formula ϕ definita dal diagramma ER, i valori dell'attributo a per l'istanza e dell'entity E saranno rappresentati dagli elementi v del dominio di interpretazione di M tali che $a(e, v) = true$

Esempio:

- Supponiamo che un diagramma ER definisca l'entity Persona avente l'attributo email di dominio stringa
- La formula ϕ definita dal diagramma conterrà occorrenze del simbolo di predicato email/1$\in \mathcal{P}$

$$\mathcal{P} = \{\ldots, \text{email}/2, \ldots\}$$

- Un modello M di ϕ definirà l'estensione del simbolo di predicato email/2

  (es: $M(\text{email}) = \{\ldots, (\alpha, \text{alpha@mymail.com}), (\alpha, \text{alpha2@mymail.com}), (\beta, \text{b@anothermail.com})\ldots\}$ rappresenta tutti e soli i valori dell'attributo email per le diverse istanze di Persona, dove Persona(α) e Persona(β))

Proposition 12. Simboli di predicato per relationship

Dato un diagramma ER, ogni **relationship** rel tra le entity $E_1, \ldots, E_n$ presente al suo interno definisce il simbolo di predicato n-ario $rel/n \in \mathcal{P}$.

In ogni **modello** M della formula ϕ definita dal diagramma ER, le istanze della relationship rel saranno rappresentati dalle n-uple $(e_1, \ldots, e_n)$ del dominio di interpretazione di M tali che $rel(e_1, \ldots, e_n) = true$ e dove $E_1(e_1), \ldots, E_n(e_n)$

Esempio:

- Supponiamo che un diagramma ER definisca la relationship "lavora" tra le entity Persona e Azienda
- La formula ϕ definita dal diagramma conterrà occorrenze del simbolo di predicato lavora/2$\in \mathcal{P}$

$$\mathcal{P} = \{\ldots, \text{lavora}/2, \ldots\}$$

- Un modello M di ϕ definirà l'estensione del simbolo di predicato lavora/2

  (es: $M(\text{lavora}) = \{\ldots, (\alpha, \gamma), (\alpha, \delta), (\beta, \delta), \ldots\}$ rappresenta tutte e sole le istanze della relationship lavora, dove Persona(α), Persona(β), Azienda(γ) e Azienda(δ))

Proposition 13. Simboli di predicato per attributi di relationship

Dato un diagramma ER, ogni **attributo** a **di una relationship** rel tra le entity $E_1, \ldots, E_n$ presente al suo interno definisce il simbolo di predicato (n+1)-ario $a/(n+1) \in \mathcal{P}$.

In ogni **modello** M della formula ϕ definita dal diagramma ER, i valori v dell'attributo a delle istanze $(e_1, \ldots, e_n)$ della relationship rel saranno rappresentati dalle (n+1)-uple $(e_1, \ldots, e_n, v)$ del dominio di interpretazione di M tali che $a(e_1, \ldots, e_n, v) = true$ e dove $E_1(e_1), \ldots, E_n(e_n)$

Esempio:

- Supponiamo che un diagramma ER definisca la relationship "lavora" tra le entity Persona e Azienda con attributo "afferenza" di dominio "data"

- La formula ϕ definita dal diagramma conterrà occorrenze del simbolo di predicato lavora/2$\in \mathcal{P}$ e del predicato afferenza/3$\in \mathcal{P}$

$$\mathcal{P} = \{\ldots, \text{lavora}/2, \text{afferenza}/3 \ldots\}$$

- Un modello M di ϕ definirà:
  - l'estensione del simbolo di predicato lavora/2

    (es: $M(\text{lavora}) = \{\ldots, (\alpha, \gamma), (\beta, \delta), \ldots\}$)
  - l'estensione del simbolo di predicato afferenza/3

    (es: $M(\text{afferenza}) = \{\ldots, (\alpha, \gamma, 03/02/1992), (\beta, \delta, 08/12/2005), \ldots\}$)

Observation 18. Estensione dei predicati e delle funzioni

Data la formula ϕ descritta da un diagramma ER, assumeremo che l'**insieme dei simboli di predicato** $\mathcal{P}$ contenga tutti gli opportuni simboli di predicato per le necessarie **relazioni matematiche** e che l'**insieme dei simboli di funzione** $\mathcal{F}$ contenga tutti gli opportuni simboli di funzione per le necessarie **operazioni matematiche**

$$\leq /2, < /2, = /2, > /2, \geq /2, \ldots \in \mathcal{P}$$

$$+/2, -/2, */2, //2, |\cdot|/2, sqrt/1, pow/2, \log_2/1, \ldots \in \mathcal{F}$$

Come nel caso del simbolo di predicato $= /2$, utilizzeremo tali simboli di predicato e di funzione aggiuntivi in modo informale (ad esempio: $x \leq y$ invece che $\leq (x, y)$)

3.3.2 Vincoli di ER in FOL

Proposition 14. Vincoli di disgiunzione di ER in FOL

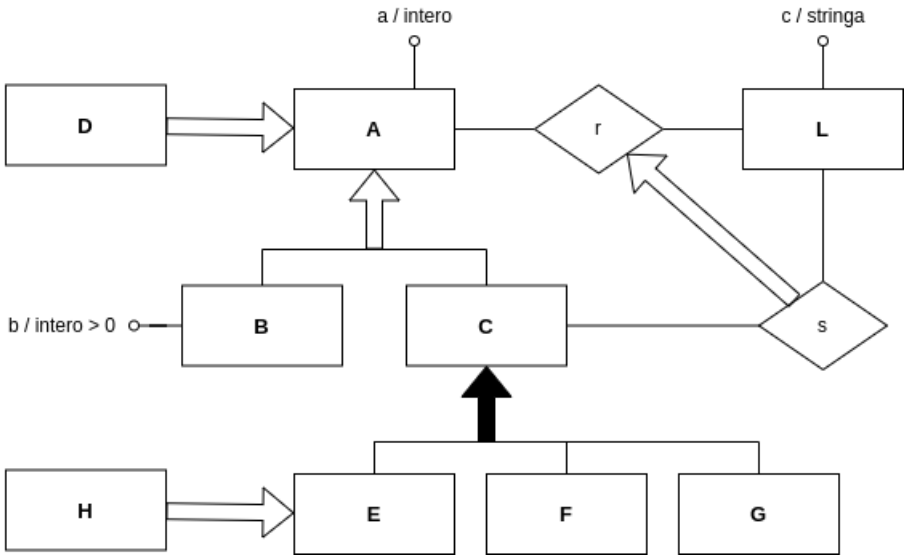
Data la formula ϕ descritta da un diagramma ER, all'interno di ϕ sono presenti delle sottoformule imponenti i **vincoli di disgiunzione** tra domini ed entity.

In particolare, al suo interno verrà imposto che:

- Due entity diverse **non appartenenti ad uno stesso albero** di relazioni is-a e/o generalizzazioni non abbiano istanze in comune
- Due entity diverse appartenenti ad uno **stesso albero** di relazioni is-a e/o generalizzazioni tali che il cammino che le congiunge passa per due entity figlie di una **stessa generalizzazione** non abbiano istanze in comune
- Due **domini diversi semanticamente disgiunti** non abbiano istanze in comune
- Una **qualunque entity** ed un **qualunque dominio** non abbiano istanze in comune

Esempio:

- Consideriamo il seguente diagramma ER



- Dai vincoli espressi all'interno della formula ϕ descritta da tale diagramma verrà imposto che:
  - Due entity diverse **non appartenenti ad uno stesso albero** di relazioni is-a e/o generalizzazioni non abbiano istanze in comune
    - $\forall x\, A(x) \rightarrow \neg L(x)$
    - $\forall x\, B(x) \rightarrow \neg L(x)$
    - ...
    - $\forall x\, H(x) \rightarrow \neg L(x)$
  - Due entity diverse appartenenti ad uno **stesso albero** di relazioni is-a e/o generalizzazioni tali che il cammino che le congiunge passa per due entity figlie di una **stessa generalizzazione** non abbiano istanze in comune
    - $\forall x\, B(x) \rightarrow \neg C(x)$
    - $\forall x\, B(x) \rightarrow \neg E(x)$
    - $\forall x\, B(x) \rightarrow \neg F(x)$
    - $\forall x\, B(x) \rightarrow \neg G(x)$
    - $\forall x\, B(x) \rightarrow \neg H(x)$
    - $\forall x\, E(x) \rightarrow \neg F(x)$
    - $\forall x\, E(x) \rightarrow \neg G(x)$
    - $\forall x\, F(x) \rightarrow \neg G(x)$
    - $\forall x\, H(x) \rightarrow \neg F(x)$
    - $\forall x\, H(x) \rightarrow \neg G(x)$
  - Due **domini diversi semanticamente disgiunti** non abbiano istanze in comune
    - $\forall x\, \text{intero}(x) \rightarrow \neg\text{stringa}(x)$
    - $\forall x\, \text{"intero} > 0\text{"}(x) \rightarrow \neg\text{stringa}(x)$

  - Una **qualunque entity** ed un **qualunque dominio** non abbiano istanze in comune
    * $\forall x\, A(x) \rightarrow \neg\text{intero}(x)$
    * ...
    * $\forall x\, L(x) \rightarrow \neg\text{intero}(x)$
    * $\forall x\, A(x) \rightarrow \neg\text{intero} > 0(x)$
    * ...
    * $\forall x\, L(x) \rightarrow \neg\text{intero} > 0(x)$
    * $\forall x\, A(x) \rightarrow \neg\text{stringa}(x)$
    * ...
    * $\forall x\, L(x) \rightarrow \neg\text{stringa}(x)$

Proposition 15. Vincoli di sottoinsieme di ER in FOL

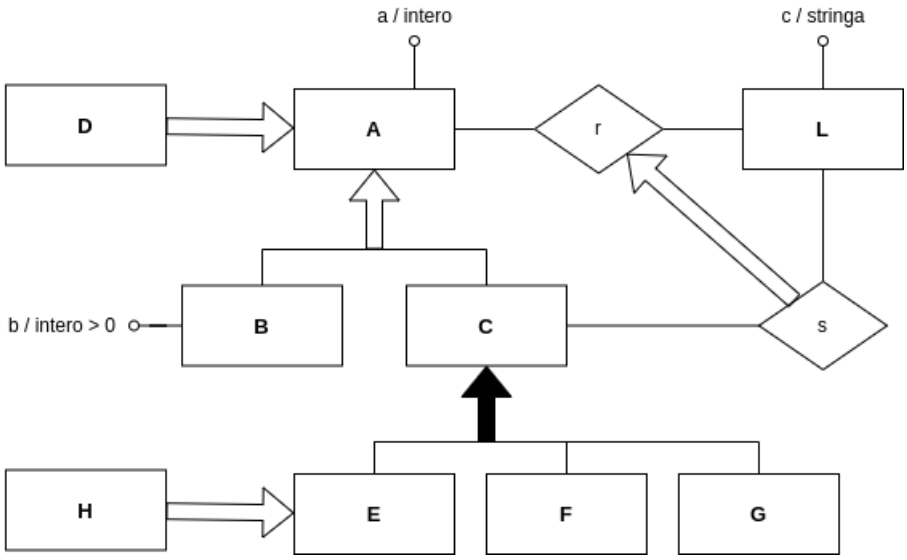
Data la formula ϕ descritta da un diagramma ER, all'interno di ϕ sono presenti delle sottoformule imponenti i **vincoli di sottoinsieme** tra entity o relationship e le loro generalizzazioni.

In particolare, al suo interno verrà imposto che:

- Le istanze di un'**entity figlia** di una generalizzazione o relazione is-a siano **anche istanze dell'entity base**
- Le istanze di una **relationship figlia** di una generalizzazione o relazione is-a siano **anche istanze della relationship base**

Esempio:

- Consideriamo ancora il diagramma ER dell'esempio precedente



- Dai vincoli espressi all'interno della formula ϕ descritta da tale diagramma verrà imposto che:
  - Le istanze di un'**entity figlia** di una generalizzazione o relazione is-a siano **anche istanze dell'entity base**
    - $\forall x\, B(x) \rightarrow A(x)$
    - $\forall x\, C(x) \rightarrow A(x)$
    - $\forall x\, D(x) \rightarrow A(x)$
    - $\forall x\, E(x) \rightarrow C(x)$
    - $\forall x\, F(x) \rightarrow C(x)$
    - $\forall x\, G(x) \rightarrow C(x)$
    - $\forall x\, H(x) \rightarrow E(x)$
  - Le istanze di una **relationship figlia** di una generalizzazione o relazione is-a siano **anche istanze della relationship base**
    - $\forall x, y\, s(x, y) \rightarrow r(x, y)$

Proposition 16. Vincolo di compleatezza di ER in FOL

Data la formula ϕ descritta da un diagramma ER, all'interno di ϕ sono presenti delle sottoformule imponenti il **vincolo di completezza** per le generalizzazioni complete.

In particolare, al suo interno verrà imposto che ogni istanza di un'**entity base** di una generalizzazione completa sia istanza di **almeno una delle entity figlie**

Attenzione: gli ulteriori vincoli di disgiunzione imporranno che tali entity figlie non possano avere istanze in comune

Esempio:

- Considerando ancora il diagramma ER dell'esempio precedente, dai vincoli esterni espressi all'interno della formula ϕ descritta da tale diagramma verrà imposto che

$$\forall x\, C(x) \rightarrow E(x) \vee F(x) \vee G(x)$$

Proposition 17. Specializzazione di domini di ER in FOL

Data la formula ϕ descritta da un diagramma ER, all'interno di ϕ sono presenti delle sottoformule descriventi i simboli di predicato per **domini specializzati** in termini dei simboli di predicato relativi ai **domini base**

In particolare, per ogni dominio specializzato dom_spec all'interno di ϕ verrà imposto che:

$$\forall x\, dom\_spec(x) \leftrightarrow (dom(x) \wedge \text{"x soddisfa il criterio di specializz."})$$

Esempio:

- $\forall x$ "intero ≥ 0"$(x) \leftrightarrow (\text{intero}(x) \land x \geq 0)$
- $\forall x\, [0, 59](x) \leftrightarrow (\text{intero}(x) \land x \geq 0 \land x \leq 59)$

Proposition 18. Tipizzazione di relationship di ER in FOL

Data la formula ϕ descritta da un diagramma ER, all'interno di ϕ sono presenti delle sottoformule imponenti che le **n-uple** dell'estensione di un predicato che definisce una **relationship** *rel* tra le entity $E_1, \ldots, E_n$ siano elementi del prodotto cartesiano $E_1 \times \ldots \times E_n$

In particolare, per ogni relationship *rel* all'interno di ϕ verrà imposto che

$$\forall e_1, \ldots, e_n\, rel(e_1, \ldots, e_n) \rightarrow E_1(e_1) \land \ldots \land E_n(e_n)$$

Esempio:

- Supponiamo che un diagramma ER definisca la relationship "lavora" tra le entity Persona e Azienda
- La formula ϕ definita dal diagramma ER conterrà la seguente sottoformula

$$\forall x, y\ \text{lavora}(x, y) \rightarrow \text{Persona}(x) \land \text{Azienda}(y)$$

Proposition 19. Tipizzazione di attributi di ER in FOL

Data la formula ϕ descritta da un diagramma ER, all'interno di ϕ sono presenti delle sottoformule imponenti che:

- Le coppie dell'estensione di un predicato che definisce i **valori di un'attributo di una o più entity** hanno come prima componente un'istanza delle entity legate e come seconda componente un'istanza del dominio dell'attributo
- Le (n+1)-uple dell'estensione di un predicato che definisce i **valori di un attributo di una o più relationship n-arie** hanno come prime n componenti una n-upla che definisce un'istanza di quella relationship e come (n+1)-esima componente un'istanza del dominio dell'attributo

Esempio:

- Supponiamo che un diagramma ER definisca la relationship "lavora" tra le entity Persona e Azienda con l'attributo "afferenza" di tipo data. Inoltre, l'entity Persona possiede un'attributo "nome" di tipo stringa
- La formula ϕ definita dal diagramma ER conterrà la seguenti sottoformule
  - $\forall e, v\ \text{nome}(e, v) \land \text{Persona}(e) \rightarrow \text{stringa}(v)$
  - $\forall e_1, \ldots, e_n, v\ \text{afferenza}(e_1, \ldots, e_n, v) \land \text{lavora}(e_1, \ldots, e_n) \rightarrow \text{data}(v)$

Proposition 20. Vincoli di cardinalità dei ruoli di ER in FOL

Data la formula ϕ descritta da un diagramma ER, all'interno di ϕ sono presenti delle sottoformule imponenti che i **vincoli di cardinalità** dei ruoli delle relationship.

In particolare, al suo interno verrà imposto che data una relationship rel tra le entity $E_1, \ldots, E_n$, per ogni ruolo $u_i, i \in [1, n]$ rispettivo di ogni entity si abbia che:

$$\forall e_i \; E_i(e_i) \rightarrow \begin{array}{l}\text{esistono almeno } min_{u_i} \text{ istanze diverse di } rel \\ \text{che hanno } e_i \text{ come } i\text{-esima componente}\end{array}$$

$$\forall e_i \; E_i(e_i) \rightarrow \begin{array}{l}\text{non esistono } max_{u_i} + 1 \text{ istanze diverse di } rel \\ \text{che hanno } e_i \text{ come } i\text{-esima componente}\end{array}$$

Nota:

I due vincoli precedenti sono stati espressi tramite linguaggio informale per via dell'estrema lunghezza della sottoformula FOL descrivente i vincoli stessi. Per completezza, di seguito vengono riportate le due sottoformule FOL:

$$\forall e_i \; E_i(e_i) \rightarrow \left(\begin{array}{c} \exists e_1^1, \ldots, e_{i-1}^1, e_{i+1}^1, \ldots, e_n^1 \ldots \exists e_1^{min_{u_i}}, \ldots, e_{i-1}^{min_{u_i}}, e_{i+1}^{min_{u_i}}, \ldots, e_n^{min_{u_i}} \\ (e_1^1, \ldots, e_{i-1}^1, e_{i+1}^1, \ldots, e_n^1) \neq (e_1^2, \ldots, e_{i-1}^2, e_{i+1}^2, \ldots, e_n^2) \\ \wedge \ldots \wedge \\ (e_1^{min_{u_i}-1}, \ldots, e_{i-1}^{min_{u_i}-1}, e_{i+1}^{min_{u_i}-1}, \ldots, e_n^{min_{u_i}-1}) \neq \\ (e_1^{min_{u_i}}, \ldots, e_{i-1}^{min_{u_i}}, e_{i+1}^{min_{u_i}}, \ldots, e_n^{min_{u_i}}) \\ \wedge \\ rel(e_1^1, \ldots, e_{i-1}^1, e_{i+1}^1, \ldots, e_n^1) \wedge \ldots \wedge \\ rel(e_1^{min_{u_i}}, \ldots, e_{i-1}^{min_{u_i}}, e_{i+1}^{min_{u_i}}, \ldots, e_n^{min_{u_i}}) \end{array} \right)$$

$$\forall e_i \; E_i(e_i) \rightarrow \neg \left(\begin{array}{c} \exists e_1^1, \ldots, e_{i-1}^1, e_{i+1}^1, \ldots, e_n^1 \ldots \exists e_1^{max_{u_i}+1}, \ldots, e_{i-1}^{max_{u_i}+1}, e_{i+1}^{max_{u_i}+1}, \ldots, e_n^{max_{u_i}+1} \\ (e_1^1, \ldots, e_{i-1}^1, e_{i+1}^1, \ldots, e_n^1) \neq (e_1^2, \ldots, e_{i-1}^2, e_{i+1}^2, \ldots, e_n^2) \\ \wedge \ldots \wedge \\ (e_1^{max_{u_i}}, \ldots, e_{i-1}^{max_{u_i}}, e_{i+1}^{max_{u_i}}, \ldots, e_n^{max_{u_i}}) \neq \\ (e_1^{max_{u_i}+1}, \ldots, e_{i-1}^{max_{u_i}+1}, e_{i+1}^{max_{u_i}+1}, \ldots, e_n^{max_{u_i}+1}) \\ \wedge \\ rel(e_1^1, \ldots, e_{i-1}^1, e_{i+1}^1, \ldots, e_n^1) \wedge \ldots \wedge \\ rel(e_1^{max_{u_i}+1}, \ldots, e_{i-1}^{max_{u_i}+1}, e_{i+1}^{max_{u_i}+1}, \ldots, e_n^{max_{u_i}+1}) \end{array} \right)$$

Proposition 21. Vincoli di cardinalità degli attributi di ER in FOL

Data la formula ϕ descritta da un diagramma ER, all'interno di ϕ sono presenti delle sottoformule imponenti che i **vincoli di cardinalità** degli attributi di entity e relationship.

In particolare, al suo interno verrà imposto che:

- Data un'entity E ed un attributo a di tale entity si abbia che:

$$\forall e\ E(e) \rightarrow \begin{array}{l}\text{esistono almeno } min_a \text{ valori diversi} \\ \text{per l'attributo } a \text{ dell'istanza } e\end{array}$$

$$\forall e\ E(e) \rightarrow \begin{array}{l}\text{non esistono } max_a + 1 \text{ valori diversi} \\ \text{per l'attributo } a \text{ dell'istanza } e\end{array}$$

- Data una relationship rel ed un attributo a di tale relationship si abbia che:

$$\forall e_1, \ldots, e_n\ rel(e_1, \ldots, e_n) \rightarrow \begin{array}{l}\text{esistono almeno } min_a \text{ valori diversi} \\ \text{per l'attributo } a \text{ dell'istanza } (e_1, \ldots, e_n)\end{array}$$

$$\forall e_1, \ldots, e_n\ rel(e_1, \ldots, e_n) \rightarrow \begin{array}{l}\text{non esistono } max_a + 1 \text{ valori diversi} \\ \text{per l'attributo } a \text{ dell'istanza } (e_1, \ldots, e_n)\end{array}$$

Nota:

I quattro vincoli precedenti sono stati espressi tramite linguaggio informale per via dell'estrema lunghezza della sottoformula FOL descrivente i vincoli stessi. Per completezza, di seguito vengono riportate le quattro sottoformule FOL:

$$\forall e\ E(e) \rightarrow \left(\begin{array}{c} \exists v_1, \ldots, v_{min_a} \\ v_1 \neq v_2 \wedge \ldots \wedge v_{min_a - 1} \neq v_{min_a} \\ \wedge a(e, v_1) \wedge \ldots \wedge a(e, v_{min_a}) \end{array} \right)$$

$$\forall e\ E(e) \rightarrow \neg \left(\begin{array}{c} \exists v_1, \ldots, v_{max_a + 1} \\ v_1 \neq v_2 \wedge \ldots \wedge v_{max_a} \neq v_{max_a + 1} \\ \wedge a(e, v_1) \wedge \ldots \wedge a(e_1, \ldots, e_n, v_{max_a} + 1) \end{array} \right)$$

$$\forall e_1, \ldots, e_n\ rel(e_1, \ldots, e_n) \rightarrow \left(\begin{array}{c} \exists v_1, \ldots, v_{min_a} \\ v_1 \neq v_2 \wedge \ldots \wedge v_{min_a - 1} \neq v_{min_a} \\ \wedge a(e, v_1) \wedge \ldots \wedge a(e_1, \ldots, e_n, v_{min_a}) \end{array} \right)$$

$$\forall e_1, \ldots, e_n\ rel(e_1, \ldots, e_n) \rightarrow \neg \left(\begin{array}{c} \exists v_1, \ldots, v_{max_a + 1} \\ v_1 \neq v_2 \wedge \ldots \wedge v_{max_a} \neq v_{max_a + 1} \\ \wedge a(e_1, \ldots, e_n, v_1) \wedge \ldots \wedge a(e_1, \ldots, e_n, v_{max_a} + 1) \end{array} \right)$$

Proposition 22. Vincoli di identificazione di ER in FOL

Data la formula ϕ descritta da un diagramma ER, all'interno di ϕ sono presenti delle sottoformule imponenti che i **vincoli di identificazione interni ed esterni** per le entity.

In particolare, al suo interno verrà imposto che:

- Non possano esistere due istanze di un'entity E aventi gli stessi valori per gli attributi $a_1, \ldots, a_n$ (**vincoli di identificazione interni**):

$$\neg \left(\begin{array}{c} \exists e_1, e_2, v_1, \ldots, v_n \; E(e_1) \wedge E(e_2) \wedge e_1 \neq e_2 \wedge \\ a_1(e_1, v_1) \wedge \ldots \wedge a_n(e_1, v_n) \wedge a_1(e_2, v_1) \wedge \ldots \wedge a_n(e_2, v_2) \end{array} \right)$$

- Non possano esistere due istanze di un'entity E aventi gli stessi valori per gli attributi $a_1, \ldots, a_n$ e legate alle stesse istanze delle altre entity con istanze coinvolte dalle relationship $r_1, \ldots, r_q$ (**vincoli di identificazione esterni**):

$$\neg \left(\begin{array}{c} \exists e_1, e_2, v_1, \ldots, v_n, f_1, \ldots, f_k, g_1, \ldots, g_p \\ E(e_1) \wedge E(e_2) \wedge e_1 \neq e_2 \wedge \\ a_1(e_1, v_1) \wedge \ldots \wedge a_n(e_1, v_n) \wedge \\ r_1(e_1, f_1, \ldots, f_k) \wedge \ldots \wedge r_q(e_1, g_1, \ldots, g_p) \wedge \\ a_1(e_2, v_1) \wedge \ldots \wedge a_n(e_2, v_n) \wedge \\ r_1(e_2, f_1, \ldots, f_k) \wedge \ldots \wedge r_q(e_2, g_1, \ldots, g_p) \end{array} \right)$$

Observation 19

Un'interpretazione `I` per una formula FOL ϕ può essere **completamente avulsa dalla realtà** ed essere comunque un modello per la formula stessa. Tale caratteristica è dovuta al grande potere di astrazione della logica, poiché la **verità** o **falsità** di una formula può essere determinata solo dopo aver definito un'interpretazione che dia la semantica dei termini atomici e delle formule atomiche e poiché il concetto di interpretazione **non è limitato** in alcun modo dal "mondo reale"

Definition 33. Semantica del mondo reale

Definiamo come **semantica del mondo reale** un'assunzione esterna alla FOL isolante un sottoinsieme di possibili interpretazioni. In particolare, **tutte le interpretazioni** della formula ϕ definita da un diagramma ER devono definire:

- L'estensione dei simboli di predicato che rappresentano domini, relazioni tra elementi di domini, semantica dei campi di domini composti
- L'estensione dei simboli di funzione che rappresentano funzioni standard tra elementi dei domini e costanti che denotano elementi di domini

in modo **consistente con la realtà**

3.3.3 Vincoli esterni in FOL

Fino ad ora, per ogni **vincolo esterno** al diagramma ER abbiamo definito un **identificatore univoco** e un'**asserzione** espressa in linguaggio naturale. Per via dell'ambiguità intrinseca al linguaggio naturale stesso, definiremo le asserzioni utilizzando la FOL.

Proposition 23. Vincoli esterni in FOL

Dato un diagramma ER, ogni vincolo esterno a tale diagramma deve essere definito all'interno del dizionario dei dati tramite:

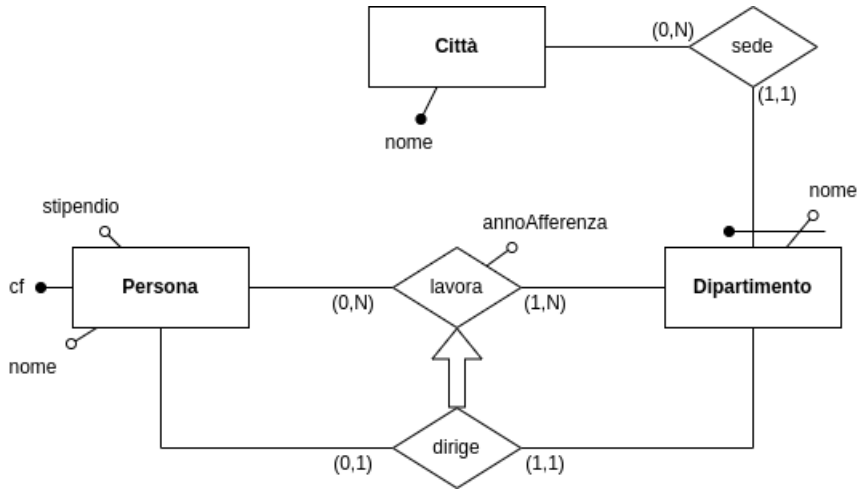
- Un'**identificatore univoco** espresso nella seguente forma:

  V.(costrutto ER).(nome vincolo)

  dove (costrutto ER) è il nome del costrutto ER (dunque entity o relationship) su cui viene applicato il vincolo e dove (nome vincolo) è un breve nome descrivente il vincolo
- Un'**asserzione** espressa in FOL

Esempio:

- Consideriamo il seguente diagramma ER



- Consideriamo quindi il seguente vincolo esterno espresso in linguaggio naturale:
  - **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*, dati:
    - Il valore $stip_{dir}$ dell'attributo stipendio di *dir*
    - Il valore $stip_p$ dell'attributo stipendio di p

    si deve avere $stip_{dir} \geq stip_p$.

- La formula FOL descrivente tale vincolo corrisponde a:

$$\forall \text{ p, dip, p, stip}_{dir}\text{, stip}_{p}\text{, } \left(\begin{array}{c} \text{dirige(dir, dip)} \wedge \\ \text{lavora(p, dip)} \wedge \\ \text{stipendio(dir, stip}_{dir}\text{)} \wedge \\ \text{stipendio(p, stip}_{p}\text{)} \end{array} \right) \rightarrow \text{stip}_{dir} \leq \text{stip}_{p}$$

- Consideriamo inoltre anche il seguente vincolo esterno:
  - **V.dirige.afferenza**: Per ogni istanza `(p: Persona, d: Dipartimento)` della relationship "dirige", l'istanza `(p: Persona, d: Dipartimento)` deve avere un valore v per l'attributo annoAfferenza tale che $v \leq$ annoCorrente -5, dove annoCorrente denota l'istanza di tipo intero rappresentante l'anno corrente
- Per rappresentare l'istante corrente nel vincolo precedente tramite cui possiamo ricavare l'anno corrente, estendiamo il vocabolario con il simbolo di costante adesso/0.

  Per la semantica del mondo reale, l'interpretazione di adesso/0 è fissata all'istanza del dominio dataora che denota l'istante corrente
- La formula FOL descrivente tale vincolo corrisponde a:

$$\forall \text{ p, dip, v, oggi, annoOggi } \left(\begin{array}{c} \text{dirige(p, dip)} \wedge \\ \text{annoAfferenza(p, dip, v)} \wedge \\ \text{data(adesso, oggi)} \wedge \\ \text{anno(oggi, annoOggi)} \end{array} \right) \rightarrow \text{v} \leq \text{annoOggi } -5$$

3.3.4 Specifiche degli use-case in FOL

Proposition 24. Operazioni di uno use-case in FOL

Dato uno use-case, un'**operazione** presente all'interno della specifica di tale use-case è una funzione avente:

- Un **input** costituito da:
  - Un **livello estensionale** dei dati formalizzabile come modello M_{in} di $\phi \wedge \zeta_1 \wedge \ldots \wedge \zeta_n$
  - Un **valore** del relativo dominio per ogni **argomento** presente nella segnatura
- Un **output** costituito da:
  - Un **valore `result`** del dominio di ritorno dom_{rit} (se definito nella segnatura)
  - Un **nuovo livello estensionale** dei dati formalizzato come un nuovo modello M_{out} di $\phi \wedge \zeta_1 \wedge \ldots \wedge \zeta_n$ nel caso in cui l'operazione abbia dei side-effect

dove ϕ è la formula FOL che definisce il diagramma ER, $\zeta_1, \ldots, \zeta_n$ sono le formule FOL che definiscono i vincoli esterni al diagramma e M_{in} ed M_{out} soddisfano la semantica del mondo reale

Proposition 25. Pre-condizioni in FOL

Dato uno use-case, un'operazione presente all'interno della specifica di tale use-case possiede delle **pre-condizioni** che formalizzano mediante una **formula FOL** π i **requisiti aggiuntivi** che il modello M_{in} di $\phi \wedge \zeta_1 \wedge \ldots \wedge \zeta_n$ e i valori degli argomenti devono soddisfare affinché l'operazione sia ben definita

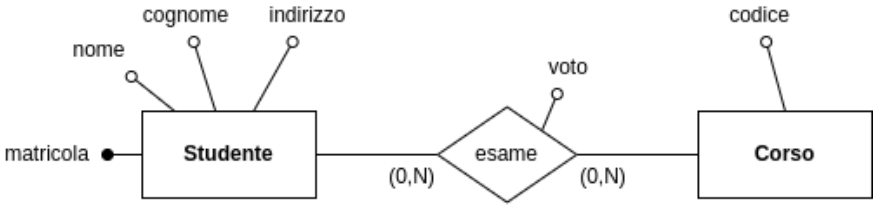
Proposition 26. Post-condizioni in FOL

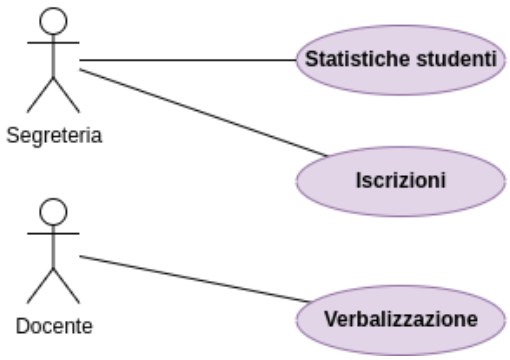
Dato uno use-case, un'operazione presente all'interno della specifica di tale use-case possiede delle **post-condizioni** che tramite una formula FOL π definiscono:

- Le differenze tra il modello M_{out} e il modello M_{in} in termini di:
  - Elementi del dominio di interpretazione $\mathcal{D}$ che esistono in M_{out} ma non in M_{in}
  - Elementi del dominio di interpretazione $\mathcal{D}$ che esistono in M_{in} ma non in M_{out}
  - n-uple di predicati che esistono in M_{out} ma non in M_{in}
  - n-uple di predicati che esistono in M_{in} ma non in M_{out}
- Il valore di ritorno `result`

Esempio:

- Consideriamo il seguente diagramma ER e il seguente diagramma degli use-case





- Consideriamo il seguente use-case:

  Verbalizzazione:

  ```
  verbalizzaEsame(s: Studente, c: Corso, v: [18, 31]):

      Pre-condizione: l'istanza s non è coinvolta con alcuna istanza
      della relationship esame con l'istanza c

      Post-condizioni: viene creata l'istanza (s: Studente, c: Corso)
      della relationship esame con valore v per l'attributo voto
  end
  ```

- Ridefiniamo quindi tale use-case utilizzando la FOL:

  Verbalizzazione:

  ```
  verbalizzaEsame(s: Studente, c: Corso, v: [18, 31]):

      Pre-condizioni:
  ```

  $$\neg \text{ esame(s,c)}$$

  ```
      Post-condizioni:
  ```

  – **Modifica del livello estensionale dei dati**: il livello estensionale dei dati al termine dell'esecuzione della funzione differisce da quello di partenza come segue:

    * **Elementi del dominio di interpretazione**: invariati
    * **Nuove n-uple di predicati:**
      · $\text{esame}(s, c)$
      · $\text{voto}(s, c, v)$
    * **N-uple di predicati che non esistono più:** nessuna

  – **Valore di ritorno**: nessuno

  ```
  end
  ```

- Definiamo quindi le ulteriori specifiche degli use case:

  Iscrizione:

  ```
  iscriviStudente(n: stringa, c: stringa, m: stringa): Studente

      Pre-condizioni:
  ```

  $$\neg \exists s \text{ Studente}(s) \wedge \text{matricola}(s, m)$$

`Post-condizioni:`

- **Modifica del livello estensionale dei dati**: il livello estensionale dei dati al termine dell'esecuzione della funzione differisce da quello di partenza come segue:
  - **Elementi del dominio di interpretazione**: un nuovo elemento α
  - **Nuove n-uple di predicati:**
    - Studente(α)
    - nome(α, n)
    - cognome(α, c)
    - matricola(α, m)
  - **N-uple di predicati che non esistono più:** nessuna
- **Valore di ritorno**:

$$\texttt{result} = \alpha$$

`cambiaIndirizzoStudente(s: Studente, i: stringa):`

`Pre-condizioni:` nessuna

`Post-condizioni:`

- Sia i_{old} il valore che, nel livello estensionale di partenza, rende vera la formula indirizzo(s, i_{old}).
- **Modifica del livello estensionale dei dati**: Il livello estensionale dei dati al termine dell'esecuzione della funzione differisce da quello di partenza come segue:
  - **Elementi del dominio di interpretazione**: invariati
  - **Nuove n-uple di predicati:**
    - indirizzo(s, i)
  - **N-uple di predicati che non esistono più:**
    - indirizzo(s, i_{old})
- **Valore di ritorno**: nessuno

`end`

Statistiche studenti:

`mediaVoti(s: Studente): reale in [18,31]`

`Pre-condizioni:`

$$\exists c \text{ esame}(s, c)$$

`Post-condizioni:`

- **Modifica del livello estensionale dei dati**: nessuna
- **Valore di ritorno**:
  - Dato
    $$C = \{(c, v) \mid \text{esame}(s, c) \land \text{voto}(s, c, v)\}$$
    si ha che:
    $$\texttt{result} = \frac{\sum_{(c,v)\in C} v}{|C|}$$

`numMedioEsami(): reale` $\geq$ `0`

`Pre-condizioni:`

$$\exists s \text{ Studente}(s)$$

`Post-condizioni:`

- **Modifica del livello estensionale dei dati**: nessuna
- **Valore di ritorno**:
  - Dati
    $$E = \{(s, c) \mid \text{esame}(s, c)\} \qquad S = \{s \mid \text{Studente}(s)\}$$
    si ha che:
    $$\texttt{result} = \frac{|E|}{|S|}$$

`end`

Observation 20. Estensioni della FOL

All'interno dell'analisi concettuale, la FOL viene estesa da:

- **Insiemi** definiti a partire da una formula FOL ϕ aperta. Ogni insieme consiste in tutte e sole le assegnazioni alle variabili libere di ϕ che rendono la formula vera.

  Se la funzione dello use-case ha dei side-effect, viene specificato se tale formula ϕ vada interpretata sul livello estensionale M_{in} o sul livello estensionale M_{out}
- **Funzioni su insiemi**, in particolare sommatorie, produttorie, cardinalità, unione, intersezione e differenza

Capitolo 4

Structured Query Language (SQL)

Basandosi sull'**architettura a tre livelli** della gestione dei DBMS, ossia

- **Livello interno**, il quale implementa le strutture fisiche di memorizzazione
- **Livello logico**, il quale fornisce un modello logico dei dati, indipendente da come sono memorizzati fisicamente (ossia il modello relazionale nel nostro caso)
- **Livello esterno**: fornisce una o più descrizioni di porzioni di interesse della base dati, indipendenti dal modello logico. Inoltre, può prevedere organizzazioni dei dati alternative e diverse rispetto a quelle utilizzate nello schema logico.

il **linguaggio SQL (Structured Query Language)** è il principale linguaggio di riferimento utilizzato per i database relazionali. Il linguaggio SQL può esser considerato l'unione di più **sottolinguaggi**, tramite cui vengono svolte le vere operazioni. In particolare, esso fornisce costrutti per operare:

- A livello logico tramite un **Data Definition Language (DDL)** per creare schemi relazionali e un **Data Manipulation Language (DML)** per interrogare e modificare i dati interni
- A livello esterno tramite ulteriori costrutti DDL per creare viste del database

Nelle seguenti sezioni, verrà utilizzata la notazione della variante SQL utilizzata dal DBMS **PostgreSQL**, nonostante le altre varianti seguano per lo più la stessa struttura.

4.1 Modello relazionale dei dati

DISCLAIMER: all'interno della seguente sezione verranno riportati solamente i concetti fondamentali e necessari già introdotti all'interno del primo modulo del corso, ossia *Basi di Dati I*.

Per tanto, tali concetti verranno trattati grossolanamente in quanto dati per assunti. Per un approfondimento maggiore, si consiglia di leggere il capitolo analogo riportato negli appunti del primo modulo.

Definition 34. Relazione matematica

Dati n insiemi $D_1, \ldots, D_n$ detti domini, una **relazione matematica** r su $D_1, \ldots, D_n$ è un sottoinsieme del loro prodotto cartesiano

$$r \subseteq D_1 \times \ldots D_n$$

In particolare, trattandosi di un insieme, ogni tupla (o ennupla) $t \in r$ è distinta dalle altre e non vi è ordinamento tra le tuple, mentre all'interno delle tuple stesse l'ordine coincide con l'ordine dei domini del prodotto cartesiano.

Observation 21. Attributi di relazioni

All'interno del modello relazionale dei dati, le relazioni matematiche vengono **estese** associando ad ogni dominio un nome detto **attributo**, il quale è unico all'interno della relazione e descrive il "ruolo" di tale dominio all'interno della relazione stessa.

Definition 35. Schema relazionale e restrizioni di tuple

Data una relazione r, viene detto **schema relazionale** di r la lista di domini $D_1, \ldots, D_n$ associati al loro attributo rispettivo $A_1, \ldots, A_n$ descriventi r, preceduti dal nome R dello schema relazionale

$$R(A_1 : D_n, \ldots, A_n : D_n)$$

Data una tupla $t \in r$ e un sottoinsieme di attributi $X \subseteq R$ componenti lo schema relazionale R, una **restrizione di** t, denotata come $t[X]$, indica il valore di t negli attributi interni ad X

Esempio:

- Consideriamo la seguente tabella rappresentante una relazione e il suo schema relazionale

Squadra Casa	Squadra Ospite	Goal Casa	Goal Ospite
Roma	Lazio	1	1
Juventus	Milan	0	1
Lazio	Roma	0	1

- Se t è la seconda tupla della tabella, allora

$$\text{t[Squadra Casa, Goal Casa]} = \text{(Juventus, 0)}$$

Definition 36. Schema di un database

Dato un insieme di schemi relazionali $R_1(X_1), \ldots, R_m(X_m)$ dove $X_1, \ldots, X_m$ sono gli insiemi di attributi di tali schemi, definiamo come tale insieme come **schema di un database**

$$\{R_1(X_1), \ldots, R_m(X_m)\}$$

Definition 37. Istanza di relazione e di database

Dato uno schema relazionale $R(X)$, definiamo come **istanza di relazione su** $R(X)$ un insieme di tuple definite su X

Analogamente, dato uno schema di un database $\{R_1(X_1), \ldots, R_m(X_m)\}$ definiamo come **istanza di database su** $\{R_1(X_1), \ldots, R_m(X_m)\}$ l'insieme $\{r_1, \ldots, r_m\}$ tale che $\forall i \in [1, m]$ si ha che r_i è un'istanza di relazione su $R_i(X_i)$

Definition 38. Valore NULL

Definiamo come **valore NULL** un valore polimorfo, ossia appartenente ad ogni dominio possibile, che denota l'**assenza di informazione**.

In particolare, il valore NULL può essere interpretato come un **valore sconosciuto, inesistente** o **temporaneamente assente**

Definition 39. Chiave e Superchiave

Dato uno schema relazionale $R(X)$, definiamo come **superchiave** un sottoinsieme $K \subseteq X$ tale che non possano esistere istanze della relazione con più di una tupla coincidente nei valori degli attributi in K.

Se non esiste un ulteriore sottoinsieme $K' \subseteq K$ rispettante la proprietà di superchiave, definiamo K' come **chiave**

Definition 40. Vincolo di integrità

Definiamo come **vincolo di integrità** una **proprietà o condizione** che deve essere soddisfatta dalle istanze delle relazioni appartenenti ad uno schema di database su cui vengono definiti tali vincoli.

I vincoli di integrità si differenziano in vincoli **intra-relazionali**, ossia interni ad una relazione, e vincoli **inter-relazionali**, ossia tra più relazioni

In particolare, individuiamo i seguenti tipi di vincoli **intra-relazionali**:

- **Vincoli di tupla**: esprimono condizioni sui valori di ciascuna tuple, indipendentemente dalle altre tuple. Un vincolo di tupla definito su un solo attributo è detto **vincolo di dominio**

  (es: voto $\geq 18 \wedge$ voto ≤ 30 è un vincolo di dominio imposto sull'attributo voto)

- **Vincolo di chiave primaria**: viene scelta una **chiave primaria** tra tutte le chiavi possibili di uno schema. Tale chiave primaria non può assumere valore NULL ed identifica univocamente una tupla, poiché nessun'altra tupla può assumere gli stessi valori all'interno della chiave. Ogni chiave primaria di un sottoschema viene indicata sottolineando gli attributi che la compongono

  (es: nello schema Studente(matricola, nome, cognome), l'attributo matricola è la chiave primaria)

Per quanto riguarda i vincoli **inter-relazionali**, invece, consideriamo in particolare il **vincolo di integrità referenziale (vincolo di foreign key)**:

- Dati due schemi relazionali R e S definiti su due relazioni r e s, ogni tupla $t \in r$ deve avere come valori del sottoinsieme di attributi $X \subseteq R$ dei valori che compaiono come valori degli attributi della chiave prima K di S.
- In altre parole, la **proiezione** su X di R deve essere un sottoinsieme della proiezione su K di S, ossia $pi_X(R) \subseteq \pi_K(S)$

  (vedere il primo modulo del corso per maggiori informazioni)
- In tal caso, definiamo X come **chiave esterna (foreign key)**

Esempio:

- Consideriamo le seguenti tre relazioni

Studente

Matricola	Nome	Cognome
1234	Mario	Rossi
0513	Luigi	Verdi
4359	Andrea	Gialli

Corso

Codice	Nome
590	Basi di Dati
591	Algebra
592	Algoritmi

Esame

Studente	Corso	Voto
1234	590	25
1234	591	22
1202	590	23

- Definendo il vincolo di integrità referenziale tra l'attributo Studente della relazione Esame e la chiave primaria Matricola della relazione Studente, tale vincolo viene violato dalla terza tupla della relazione Esame, poiché non esiste alcuna tupla interna alla relazione Studente avente 1202 come valore di chiave primaria
- Definendo il vincolo di integrità referenziale tra l'attributo Corso della relazione Esame e la chiave primaria Codice della relazione Corso, tale vincolo viene rispettato da ogni tupla

Proposition 27. Azioni compensative per l'eliminazione

Date due relazioni con schemi R e S e istanze r ed s, se esiste un vincolo di integrità referenziale tra una chiave esterna $K \subseteq R$ e una chiave primaria $K' \subseteq S$ e viene **eliminata una tupla** $t \in s$, il DBMS (Database Management System) gestirà le **violazioni del vincolo di foreign key** situazione in tre possibili modi:

- Verrà impedita l'operazione se esiste almeno una tupla in r avente in K gli stessi valori di $t[K']$
- In tutte le tuple in r aventi in K gli stessi valori di $t[K']$ verranno sostituiti tali valori in K con dei valori NULL
- Tutte le tuple in r aventi in K gli stessi valori di $t[K']$ verranno eliminate, ripetendo il processo ricorsivamente in ogni schema in cui verrà nuovamente violato il vincolo di foreign key a seguito di tale operazione (**eliminazione a cascata**)

4.2 Creazione di database, schemi e tabelle

Proposition 28. Creazione di database, schemi, tabelle e domini

Tramite il Data Defition Language, in SQL è possibile definire la struttura dell'intero database. In particolare, vengono forniti costrutti per:

- Creare un database:

```
create database nome_database [opzioni];
```

- Creare uno schema, ossia un namespace (una sorta di "sottoinsieme") del database:

```
create schema nome_schema [opzioni];
```

- Creare una tabella:

```
create table [nome_schema].<nome_tabella> (
    <nome_attributo_1> <dominio_a1> [vincoli_dominio_a_1],
    <nome_attributo_1> <dominio_a1> [vincoli_dominio_a_1],
    ...
    <nome_attributo_1> <dominio_a1> [vincoli_dominio_a_1],
    [altri_vincoli_intra-relazionali]
    [altri_vincoli_inter-relazionali]
);
```

(le le parentesi quadre indicano parametri opzionali)

Observation 22. Domini predefiniti in SQL

Per quanto riguarda i **domini predefiniti** del linguaggio SQL, troviamo:

- **Domini numerici**, come:
  - `integer`, `smallint`, ...
  - `numeric(precision, scale)`, `decimal(precision, scale)`, ...
  - `float(precision)`, `real`, `double precision`, ...
- **Domini stringhe**, come:
  - `character [varying] (max_length)` *(abbreviabile con* `char` *o* `varchar`*)*
  - `text`, ...
- **Domini temporali**, come:
  - `date`, `time` e `timestamp` *(corrispondente all'unione di* `date` *e* `time`*)*
  - `interval` *(corrispondente ad un intervallo temporale)*
- **Dominio booleano**, ossia `Boolean`
- ...

Esempi:

- Definiamo la tabella Corso come segue, indicando che `codice` sia la sua **chiave primaria**

```
create table Corso (
    codice integer not null,
    nome varchar(100) not null,
    aula varchar(100) not null,
    primary key (codice)
);
```

- Definiamo la tabella Docente come segue, indicando che il campo `corso` sia una **chiave esterna** facente riferimento alla chiave primaria della tabella Corso

```
create table Docente (
    matricola integer not null,
    nome varchar(100) not null,
    cognome varchar(100) not null,
    corso integer not null,
    primary key (matricola)
    foreign key (corso) references Corso(codice)
);
```

- Definiamo la tabella Impiegato, indicando che il campo `stipendio` abbia come **valore di default** il valore 0 tramite la keyword `default` e imponendo su di esso il **vincolo di dominio** `stipendio` ≥ 0 tramite la keyword `check`

```
create table Impiegato(
    nome varchar(100) not null,
    cognome varchar(100) not null,
    stipendio integer default 0
                check (stipendio >= 0),
    ...
);
```

- Definiamo la tabella Studente come segue, indicando con una notazione alternativa che il campo `matricola` sia chiave primaria (implicando anche il vincolo `not null`) e indicando che il campo `cf` e la tripla di campi `(cognome, nome, nascita)` siano due ulteriori chiavi non primarie di Studente tramite la keyword `unique`

```
create table Studente(
    matricola integer primary key,
    cf char(16) not null,
    nome varchar(100) not null,
    cognome varchar(100) not null,
    nascita date,
    unique (cf),
    unique (cognome, nome, nascita)
);
```

Observation 23

Di seguito, una lista delle keyword viste negli esempi precedenti:

- `attributo` **`not null`** indica che tale attributo non possa assumere valore null
- `attributo` **`default`** `valore` indica che tale attributo assuma il valore di default indicato se non viene specificato
- `attributo` **`check`**`(vincolo)` indica che tale attributo non possa violare il vincolo di integrità imposto
- **`unique`** `(lista_attributi)` indica che tali attributi siano una chiave
- `attributo` **`primary key`** o **`primary key`** `(lista_attributi)` indica che tale/i attributo/i sia/siano la chiave primaria (include `not null`)
- **`foreign key`** `(lista_attributi) references Tabella(chiave)` indica che tali attributi siano una chiave esterna facente riferimento alla chiave della tabella indicata

Observation 24. Modifica ed eliminazione di tabelle

Per modificare la struttura delle tabelle, il linguaggio SQL fornisce costrutti utilizzanti la keyword `alter`, ad esempio:

- `alter table <nome_tabella> add column ...`
- `alter table <nome_tabella> drop column ...`
- `alter table <nome_tabella> add constraint ...`
- ...

Analogamente, vengono forniti costrutti per la cancellazione di tabelle, schemi o database tramite la keyword `drop`:

- `drop table <nome_tabella>`
- `drop schema <nome_schema>`
- `drop database <nome_database>`

Per maggiori dettagli, è consigliata la visione della documentazione del linguaggio SQL e del DBMS utilizzato

Proposition 29. Dominio specializzato

Un **dominio specializzato in SQL** definisce un sottoinsieme di un dominio SQL predefinito tramite il comando

```
create domain <nome_dominio> as <dominio_base>
    [valore_default]
    [vincoli]
;
```

Attenzione: i vincoli vengono definiti su un campo "immaginario" indicato con la keyword `value`

Esempio:

- Definiamo un dominio specializzato `voto` come:

```
create domain voto as integer
    default 0
    check (value >= 18 and value <= 30)
;
```

Proposition 30. Dominio enumerativo e dominio record

Un **dominio enumerativo in SQL** definisce un insieme finito di n valori, ognuno identificato da una etichetta tramite la keyword `type`:

```
create type <nome_dominio> as enum(
    'valore_1', ..., 'valore_n'
);
```

Un **dominio record in SQL**, invece, definisce un dominio di m tuple i cui campi corrispondono ad un dominio associato, sempre tramite la keyword `type`:

```
create type <nome_dominio> as (
    <campo_1> <dominio_1>,
    ...
    <campo_m> <dominio_m>
);
```

Attenzione: le etichette dei domini enumerativi non sono elementi di dominio stringa, bensì dei semplici identificatori

Esempio:

- Definiamo un dominio enumerativo `continente` come:

```
create type continente as enum(
    'Africa', 'America Nord', 'America Sud',
    'Antartica', 'Asia', 'Europa', 'Oceania'
);
```

- Definiamo un dominio `indirizzo` come:

```
create type indirizzo as (
    citta varchar(100),
    via varchar(200),
    civico integer
);
```

Proposition 31. Valori progressivi

Un **valore progressivo** in SQL è una funzionalità fornita dal DBMS permettente di affidare a quest'ultimo il compito di assegnare valori diversi (tipicamente progressivi) per un campo a seguito della creazione di un'entrata

```
create sequence <nome_valore_progressivo>;

create table <nome_tabella>(
    <attributo_1> <dominio_1> default nextval(
        'nome_valore_progressivo'
    )
    ...
);
```

Esempio:

- Definiamo la tabella Prenotazione come segue, utilizzando un valore progressivo per la chiave primaria (notazione PostgreSQL)

```
create sequence Prenotazione_id_seq;

create table Prenotazione (
    id integer default nextval(
        "Prenotazione_id_seq"
    ) primary key,
    istante timestamp not null
);
```

- Analogamente, la sintassi MySQL prevede la keyword `auto_increment` per la stessa funzionalità

```
create table Prenotazione (
    id integer primary key auto_increment,
    istante timestamp not null
);
```

4.3 Interrogazioni su tabelle

Proposition 32. Interrogazioni su tabelle

Per effettuare una **query** su una tabella, il linguaggio SQL fornisce il costrutto:

```
select <attr_1>, ..., <attr_n>
    from <tabella_1>, ..., <tabella_m>
    [where <condizione>]
```

il quale è **quasi** equivalente a:

$$\pi_{\text{attr\_1, ..., attr\_n}}(\sigma_{\text{condizione}}(\text{tabella\_1} \times ... \times \text{tabella\_m}))$$

Nota: nello svolgere le query, il DBMS non effettua realmente il prodotto cartesiano tra tabelle (poiché troppo costoso), ma effettua alcune ottimizzazioni.

Observation 25. Select distinct

La proiezione effettuata dalla clausola `select` all'interno di una query non elimina tuple duplicate, implicando che, se esistono due tuple uguali, **entrambe verranno mantenute**.

Per eliminare le tuple duplicate, è necessario utilizzare la keyword `select distinct`, rendendo dunque la query SQL **effettivamente uguale** alla query in algebra relazionale

Esempio:

1. • Consideriamo la seguente tabella:

Persona

Nome	Cognome	Età
Mario	Rossi	47
Clemente	Verdi	63
Mario	Rossi	42
Lorenzo	Gialli	34

• La seguente query

```
select distinct Persona.nome, Persona.cognome, Persona.eta
    from Persona
    where Persona.eta > 40 and Persona.eta < 45
```

corrisponde esattamente a

$$\pi_{\text{nome, cognome}}(\sigma_{\text{età}>40 \wedge \text{età}<45}(\text{Persona}))$$

2. • Consideriamo le seguenti tabelle:

Officina

ID	Nome	Indirizzo
1	FixIt	via delle Spighe 4
2	CarFix	via delle Betulle
3	MotorGo	piazza Turing 1

Riparazione

ID	Officina	Veicolo
1	2	HK242BW
2	2	AA662XQ
3	1	HK242BW

• La seguente query

```
select Officina.ID, Officina.indirizzo
    from Officina, Riparazione
    where Officina.nome = Riparazione.officina
        and Riparazione.veicolo = 'HK242BW'
```

corrisponde (quasi, per l'assenza della keyword `distinct`) a

$$\pi_{\text{Off.ID, Off.indirizzo}}(\sigma_{\text{Off.nome} = \text{Rip.officina} \wedge \text{Rip.veicolo} = \text{'HK242BW'}}(\text{Off} \times \text{Rip}))$$

Observation 26. Ambiguità tra gli attributi

Se non vi è ambiguità tra i nomi degli attributi, all'interno della clausola `select` e della clausola `where` può essere **omesso** il nome della tabella

Esempio:

- Le due query dell'esempio precedente sono equivalenti alle seguenti due query

```
select distinct nome, cognome, eta
    from Persona
    where eta > 40 and eta < 45

select Officina.ID, indirizzo
    from Officina, Riparazione
    where nome = officina
        and veicolo = 'HK242BW'
```

Observation 27. Carattere speciale *

All'interno della clausola `select`, può essere utilizzato il **carattere speciale *** al posto di specificare **tutti i campi della tabella**

Esempio:

- Riprendendo la query precedente sulla tabella Persona, la seguente query è equivalente alla precedente

```
select distinct *
    from Persona
    where eta > 40 and eta < 45
```

Observation 28. Ridenominazione di attributi

All'interno della clausola `select` è possibile specificare un alias a seguito del nome di ogni attributo utilizzando la keyword `as` al fine di **ridenominarlo**

Esempio:

- La seguente query effettua una ridenominazione dell'attributo `Riparazione.officina` in `ID_officina`

```
select officina as ID_officina
    from Riparazione
```

Observation 29. Aliasing di tabelle

All'interno della clausola `from` è possibile specificare un **alias** per le tabelle utilizzate a seguito del nome di ogni tabella utilizzando la keyword `as`

```
...
from <tabella_1> as <alias_1>, ..., <tabella_m> as <alias_m>
...
```

oppure omettendo anche la keyword `alias`

```
...
from <tabella_1> <alias_1>, ..., <tabella_m> <alias_m>
...
```

Gli alias possono essere utilizzati all'interno delle altre clausole della query **al posto del nome della tabella**.

Esempio:

- Riprendendo la query precedente sulle tabelle Officina e Riparazione, la seguente query è equivalente alla precedente

```
select o.ID, indirizzo
    from Officina as o, Riparazione as r
    where nome = officina
        and veicolo = 'HK242BW'
```

Proposition 33. Condizioni is null e is not null

All'interno della clausola `where`, possono essere inserite le keyword `is null` e `is not null`, le quali, rispettivamente, selezioneranno le tuple il cui campo indicato è impostato su NULL o non è impostato su NULL

Esempio:

- La seguente query restituisce le tuple la cui data di nascita non è impostata su null

```
select * from Persona
where nascita eta is not null
```

Proposition 34. Condizione like

All'interno della clausola `where`, può essere inserita la keyword `like`, la quale selezionerà le tuple il cui campo indicato rispetta il **pattern** indicato a seguito della keyword stessa

Per definire i pattern, vengono utilizzati i **caratteri speciali % e _**, i quali, rispettivamente, indicano di verificare la corrispondenza con una qualsiasi stringa ed un qualsiasi carattere

Esempi:

- La seguente query restituisce le tuple il cui cognome inizia per 'R'

```
select * from Persona
where cognome like 'R%'
```

- La seguente query restituisce le tuple il cui cognome inizia per 'R' ed ha un carattere 's' come terzo carattere

```
select * from Persona
where cognome like 'R_s%'
```

- La seguente query restituisce i cognome contenenti 'r'

```
select * from Persona
where cognome like '%r%'
```

Proposition 35. Ordinamento delle tuple

Tramite la keyword `order by` è possibile **ordinare** le tuple restituite da una query per valore crescente o decrescente in base all'attributo specificato

```
select ...
    from ...
    where ...
    order by <attributo_1> asc | desc, ..., <attributo_n> asc | desc
```

Esempio:

- Riprendendo la precedente tabella Officine, la seguente query restituisce le tuple della tabella per nome crescente (ossia in ordine lessicografico)

```
select * from Officina
    where ...
    order by nome asc
```

4.3.1 Operatori aggregati e Raggruppamenti

Definition 41. Operatori aggregati

All'interno delle clausole `select` e `where` è possibile utilizzare degli **operatori aggregati**, ossia delle funzioni in grado di calcolare un valore richiesto utilizzando tutte le tuple selezionate dalla query

In particolare, consideriamo i seguenti principali operatori aggregati:

- **`count`**`(*)`, il quale restituisce il numero di tuple restituite della query
- **`count`**`(<nome_attributo>)`, il quale restituisce il numero di valori non NULL (e non distinti) dell'attributo indicato presenti nelle tuple restituite della query
- **`count(distinct`** `<nome_attributo>)`, il quale restituisce il numero di valori non NULL (e distinti) dell'attributo indicato presenti nelle tuple restituite della query
- **`sum`**`(<nome_attributo>)`, il quale restituisce la somma dei valori dell'attributo indicato presenti nelle tuple restituite dalla query (solo per domini numerici)
- **`avg`**`(<nome_attributo>)`, il quale restituisce la media dei valori dell'attributo indicato presenti nelle tuple restituite dalla query (solo per domini numerici o temporali)
- **`min`**`(<nome_attributo>)`, il quale restituisce il minimo tra i valori dell'attributo indicato presenti nelle tuple restituite dalla query (solo per domini ordinabili)
- **`max`**`(<nome_attributo>)`, il quale restituisce il massimo tra i valori dell'attributo indicato presenti nelle tuple restituite dalla query (solo per domini ordinabili)

(gli operatori `sum`*,* `avg`*,* `min` *e* `max` *ignorano i valori NULL)*

Esempi:

- Consideriamo la seguente tabella

Impiegato

Nome	Cognome	Eta	Stipendio
Mario	Rossi	47	1200
Giulia	Verdi	35	NULL
Anna	Gialli	55	1300
Fabio	Rossi	44	1200

- La seguente query restituirà una singola tupla contenente un attributo `count(*)` avente valore 4

```
select count(*) from Impiegato
```

- La seguente query restituirà una singola tupla contenente un attributo `count(stipendio)` avente valore 3

```
select count(stipendio) from Impiegato
```

- La seguente query restituirà una singola tupla contenente un attributo `count(distinct stipendio)` avente valore 2

```
select count(distinct stipendio) from Impiegato
```

- La seguente query restituirà una singola tupla contenente lo stipendio minimo, lo stipendio massimo e lo stipendio medio

```
select
    min(stipendio) as minStip,
    max(stipendio) as maxStip,
    avg(stipendio) as avgStip
from Impiegato
```

Proposition 36. Clausole group by ed having

Gli operatori aggregati possono essere applicati a **partizioni delle tuple** aggiungendo la clausola `group by` ad una query.

```
select <attr_1>, ..., <attr_k> <aggr_1(...)>, ..., <aggr_n(...)>
from <tabella_1>, ..., <tabella_m>
where <condizione>
group by <attr_a>, <attr_b>, ..., <attr_g>
having <condizione_sui_gruppi>
```

La query viene valutata come segue:

1. Viene prima eseguita la normale query `select ... from ... where`
2. Vengono partizionate le tuple risultanti mettendo nello stesso **gruppo** quelle coincidenti nei valori di tutti gli attributi specificati nella clausola `group by`
3. Per ogni gruppo viene calcolato indipendentemente il valore delle funzioni aggregate
4. Viene restituita una tupla per ogni gruppo con i valori di (alcuni tra) gli attributi della clausola `group by` e i valori delle funzioni aggregate per il singolo gruppo
5. Le tuple per i gruppi che non soddisfano le condizioni imposte nella clausola `having` vengono scartate

Esempio:

- Consideriamo le seguenti due tabelle

Persona

ID	Nome	Eta
1001	Luca	45
1002	Anna	48
1003	Giulia	35
1004	Maria	45
1005	Antonio	48

GenitoreFiglio

Genitore	Figlio
1001	1005
1001	1003
1004	1003
1002	1005

- Consideriamo quindi la seguente query restituente i nomi delle persone con figli e con età maggiore o uguale a 45 anni, assieme ai nomi dei loro figli

```
select g.ID as gID, g.nome as genitore, f.nome as figlio
from Persona g, GenitoreFiglio gf, Persona f
where g.ID = gf.genitore and gf.figlio = f.ID and g.eta >= 45
```

gID	Genitore	Figlio
1001	Luca	Antonio
1001	Luca	Giulia
1002	Anna	Antonio
1004	Maria	Giulia

- Consideriamo invece la seguente query restituente i nomi delle persone con figli e con stipendio maggiore o uguale a 45 anni, assieme al numero dei loro figli

```
select g.ID as gID, g.nome as genitore, count(f.ID) as nFigli
from Persona g, GenitoreFiglio gf, Persona f
where g.ID = gf.genitore and gf.figlio = f.ID and g.eta >= 45
group by g.ID, g.nome
```

gID	Genitore	nFigli
1001	Luca	2
1002	Anna	1
1004	Maria	1

- Infine, consideriamo la seguente query restituente i nomi delle persone con figli e con stipendio maggiore o uguale a 45 anni ed almeno due figli, assieme al numero dei loro figli

```
select g.ID as gID, g.nome as genitore, count(f.ID) as nFigli
from Persona g, GenitoreFiglio gf, Persona f
where g.ID = gf.genitore and gf.figlio = f.ID and g.eta >= 45
group by g.ID, g.nome
having count(f.ID) >= 2
```

gID	Genitore	nFigli
1001	Luca	2

Observation 30. Omogeneità della target list

Se all'interno della clausola `select` viene utilizzato un'operatore aggregato, non potranno essere utilizzati dei semplici attributi all'interno della target list, ma solo ulteriori operatori aggregati (a meno che non venga usata anche la clausola `group by`)

Esempio:

- La seguente query è errata, poiché la proiezione sull'attributo `nome` restituirebbe un insieme di tuple, mentre l'operatore aggregato `avg` restituirebbe una singola tupla, andando in conflitto vicendevolmente

```
select nome, avg(stipendio)
    from Impiegato
```

4.3.2 Sotto-query e Viste

Proposition 37. Sotto-query

All'interno delle clausole `from` e `where`, possono essere inserite delle **sotto-query**, ottenendo una valutazione **stratificata**, dando precedenza alle query più interne

```
select ...
from <tabella_1>, ..., (
    select ...
    )
where ..., <attr> = [any | all](select ...)
```

In particolare, all'interno delle clausola `from` è possibile utilizzare le keyword `any` e `all` per valutare una condizione su **qualcuna** od **ogni** tupla della sotto-query e le keyword `exists` e `not exists` per valutare la query esterna **solo se esiste** almeno una tupla o **non esiste alcuna** tupla all'interno della sotto-query.

Attenzione: è possibile omettere le keyword `any`, `all`, `exists` e `not exists` se e solo se la sotto-query restituirà **una ed una sola tupla**, altrimenti la query verrà considerata invalida

Esempi:

- La seguente query restituisce la media dei redditi totali dei figli di ogni madre

```
select avg(q.tot_reddit_figli) as avg_redd_tot_figli
from (
    select m.nome as madre, sum(f.reddito) as tot_reddito_figli
    from Maternita m, Persona f
    where m.figlio = f.nome
    group by m.nome
) as q
```

- La seguente query restituisce nome e reddito dei padri di persone con reddito maggiore di 1000

```
select distinct nome, reddito
from Persona p
where p.nome = any (
    select pat.padre
    from Paternita pat, Persona f
    where f.nome = pat.figlio and f.reddito > 20
)
```

- Le seguenti due query sono equivalenti in quanto entrambe restituiscono il nome delle persone con reddito maggiore di tutte le persone di età inferiore ai 30 anni

```
select nome from Persona p
where p.reddito > all (
    select reddito from Persona where eta < 30
)

select nome from Persona p
where p.reddito > (
    select max(reddito) from Persona where eta < 30
)
```

Observation 31

Una sotto-query **può fare riferimento** agli attributi presenti nella **query esterna**

Tuttavia, una query **non può far riferimento** agli attributi presenti in una sua **sotto-query**

Esempio:

- La seguente query restituisce i dati delle persone con almeno un figlio

```
select * from Persona p
    where exists (
        select * from Paternita where padre = p.nome
    )
    or exists (
        select * from Maternita where madre = p.nome
    )
```

Definition 42. Viste su dati

Una **vista** è una **tabella virtuale** le cui tuple sono calcolate a partire da un'interrogazione su altre tabelle e/o viste.

```
create view <nome_vista> as
    select ...
```

Ogni vista può essere **richiamata da altre query**, le quali possono utilizzarla come se fosse una tabella reale.

Esempio:

- Creiamo la seguente vista costituente una tabella virtuale Genitori

```
create view Genitori as
    select mat.figlio as persona, mat.madre as madre, pat.padre
       as padre
    from Maternita mat, Paternita pat
    where mat.figlio == pat.figlio
```

- Una volta creata tale vista, possiamo utilizzarla come se fosse una qualsiasi tabella

```
select * from Genitori
    where ...
```

Observation 32

Il contenuto di ogni vista viene **calcolato al momento della richiesta** e non viene memorizzato in modo persistente, pertanto non è possibile invocare le keyword `insert`, `delete` ed `update` su di esse.

4.3.3 Operatori insiemistici

Proposition 38

Il linguaggio SQL fornisce tre costrutti costrutti corrispondenti agli **operatori insiemistici** dell'algebra relazionale:

- **Unione**: restituisce le tuple presenti in una qualsiasi delle due query indicate. Tramite la keyword `distinct` può essere specificato di eliminare i duplicati (comportamento di default), mentre tramite la keyword `all` può essere specificato di non eliminarli.

```
<query_1>
union [all | distinct]
<query_2>
```

- **Intersezione**: restituisce le tuple presenti in entrambe le due query indicate.

```
<query_1>
intersect
<query_2>
```

- **Differenza**: restituisce le tuple presenti nella prima query ma non nella seconda query.

```
<query_1>                          <query_1>
except              oppure         minus
<query_2>                          <query_2>
```

Observation 33

Come nell'algebra relazionale, per poter utilizzare gli operatori insiemistici, le tuple restituite dalle due query devono essere **compatibili in unione**, ossia devono avere la stessa quantità di attributi ed ogni i-esimo attributo di entrambi gli insiemi di tuple deve essere dello stesso dominio (non importa il nome degli attributi)

Observation 34

All'interno delle sotto-query **non è possibile utilizzare** gli operatori insiemistici

4.3.4 Join esplicito, Join naturale e Outer Join

Observation 35. Join esplicito

Oltre alla notazione già vista

```
select ...
from <tabella_1>, <tabella_2>
where <condizione join> and <condizione query>
```

il linguaggio SQL prevede una **sintassi esplicita equivalente** per l'operazione di join:

```
select ...
from <tabella_1> join <tabella_2>
    on <condizione join>
where <condizione query>
```

Attenzione: tali costrutti corrispondono al **theta join** dell'algebra relazione

Proposition 39. Join naturale

Il linguaggio SQL fornisce un costrutto corrispondente al **join naturale** dell'algebra relazionale:

```
select ...
from <tabella_1> natural join <tabella_2>
where ...
```

Proposition 40. Outer join

Oltre al join naturale e al theta join, il linguaggio SQL fornisce tre costrutti inerenti al concetto di **outer join**, ossia una versione meno restringente di join:

```
select ...
from <tabella_sx> left | right | full outer join <tabella_dx>
where <condizione join> and <condizione query>
```

- **Left outer join**: tutte le tuple della tabella **sinistra** partecipano al join.

  Per ogni tupla della tabella sinistra, si ha che:

  - Se esiste **una tupla o più tuple** all'interno della tabella **destra** rispettanti la condizione del join assieme alla tupla sinistra considerata, esse verranno joinate, come in un normale join.
  - Se **non esiste alcuna tupla** all'interno della tabella **destra** rispettante la condizione del join assieme alla tupla sinistra considerata, quest'ultima verrà joinata assieme ad una **"finta tupla"** avente valore NULL per ogni attributo della tabella destra

- **Right outer join**: analogo al left outer join, ma la tabella sinistra e destra assumono ruoli invertiti
- **Full outer join**: la tabella sinistra e destra assumono entrambi i ruoli (può essere visto come l'utilizzo dell'operatore unione tra il left outer join e il right outer join).

Esempi:

- Consideriamo le seguenti due tabelle

Azienda

<u>ID</u>	Nome
1	DELL
2	HP
3	IBM
4	Microsoft
5	Apple

Impiegato

<u>ID</u>	Nome	Azienda
1001	Marco	2
1002	Mario	3
1003	Giulia	NULL
1004	Lisa	6
1005	Paolo	3

- Le tuple restituite dal seguente left outer join corrispondono a

```
select *
from Azienda left outer join Impiegato
where Azienda.ID = Impiegato.Azienda
```

Azienda.ID	Azienda.Nome	Impiegato.ID	Impiegato.Nome	Azienda
1	DELL	NULL	NULL	NULL
2	HP	1001	Marco	2
3	IBM	1002	Mario	3
3	IBM	1005	Paolo	3
4	Microsoft	NULL	NULL	NULL
5	Apple	NULL	NULL	NULL

- Le tuple restituite dal seguente right outer join corrispondono a

```
select *
from Azienda right outer join Impiegato
where Azienda.ID = Impiegato.Azienda
```

Azienda.ID	Azienda.Nome	Impiegato.ID	Impiegato.Nome	Azienda
2	HP	1001	Marco	2
3	IBM	1002	Mario	3
NULL	NULL	1003	Giulia	NULL
NULL	NULL	1004	Lisa	6
3	IBM	1005	Paolo	3

- Le tuple restituite dal seguente full outer join corrispondono a

```
select *
from Azienda full outer join Impiegato
where Azienda.ID = Impiegato.Azienda
```

Azienda.ID	Azienda.Nome	Impiegato.ID	Impiegato.Nome	Azienda
1	DELL	NULL	NULL	NULL
2	HP	1001	Marco	2
3	IBM	1002	Mario	3
3	IBM	1005	Paolo	3
4	Microsoft	NULL	NULL	NULL
5	Apple	NULL	NULL	NULL
NULL	NULL	1003	Giulia	NULL
NULL	NULL	1004	Lisa	6

Observation 36

A differenza dell notazione non esplicita del join naturale, all'interno dei costrutti `natural join`, e `left | right | full outer join` **non è possibile effettuare l'aliasing** delle tabelle coinvolte nel join

4.3.5 Espressioni e funzioni

Observation 37. Espressioni e funzioni

All'interno dei costrutti `select` possono essere utilizzate **espressioni matematiche** e **funzioni** (matematiche, su stringhe, ...)

Tra le funzioni principali troviamo:

- **Funzioni matematiche**:

  abs(value), **exp**(value), **log**(value), **log**(base, value), **power**(base, value), **sqrt**(value), **floor**(value), **sin**(value), ...

- **Funzioni su stringhe**:

  upper(string), **lower**(string), **initcap**(string), **ltrim**(string, trimlist), **rtrim**(string, trimlist), **replace**(string, target, replacement), **substring**(string, pos, len), **translate**(string, fromlist, tolist), **length**, (string), **to_char**(string), **position**(string, substring), ...

- **Funzioni per pattern matching**, ad esempio `similiar` per le regular expression (regEx)
- **Funzioni per i domini temporali**
- ...

Esempi:

- La seguente query restituisce tuple composte dal campo `nome`in cui il carattere iniziale del viene modificato in maiuscolo e dal risultato di un'espressione matematica (senza alcun senso effettivo)

```
select
    initcap(nome) as nome,
    sqrt(eta)-round(reddito/100) as nonHaSenso
from Impiegato
where eta < round(sqrt(850)+length(nome))
```

- La seguente query restituisce tuple composte dall'età degli impiegati raggruppati per età e il numero di tuple per gruppo sommato al reddito massimo per gruppo (ancora una volta, esempio senza senso effettivo)

```
select eta, count(*)+max(reddito) as nonHaSenso
from Impiegato group by eta
```

- La seguente query formatta la data indicata secondo il formato indicato, restituendo la tupla ('21 May 2013')

```
select to_char(date('2013-05-21'), 'DD FMMonth YYYY') as result
```

4.4 Inserimento, Cancellazione e Modifica di tuple

Proposition 41. Inserimento, cancellazione e modifica di tuple

Per **inserire**, **cancellare** o **modificare** le tuple di una tabella, il linguaggio SQL fornisce i seguenti costrutti:

- **Inserimento**: vengono aggiunte m tuple alla tabella indicata aventi gli attributi specificati

```
insert into <nome_tabella>(<attributo_1>, ..., <attributo_n>)
    values
    (<val_1_tup_1>, ..., <val_n_tup_1>),
    ...
    (<val_1_tup_m>, ..., <val_n_tup_m>)
;
```

- **Cancellazione**: vengono rimosse tutte le tuple della tabella indicata rispettanti la condizione data (ossia tutte le tuple generate da $\sigma_{\text{condizione}}(\text{nome\_tabella})$)

```
delete from <nome_tabella>
    where <condizione>
;
```

- **Modifica**: vengono modificate tutte le tuple della tabella indicata rispettanti la condizione data (ossia tutte le tuple generate da $\sigma_{\text{condizione}}(\text{nome\_tabella})$)

```
update <nome_tabella>
    set <attributo_1> = <val_1>, ..., <attributo_n> = <val_n>
    where <condizione>
;
```

Observation 38. Inserimento di default

Se in un costrutto `insert` non viene specificato uno o più attributi definiti all'interno di una tabella, verranno inseriti i **valori di default** per tali attributi all'interno delle nuove tuple

Esempio:

- Data la tabella `Impiegato(codice, nome, cognome, stipendio)` dove il valore di default per l'attributo stipendio è 0, la seguente query inserirà la tupla (1234, 'Mario', 'Rossi', 0) all'interno della tabella

```
insert into Impiegato(codice, nome, cognome)
    values (1234, 'Mario', 'Rossi')
```

Observation 39. Modifica in default

All'interno di un costrutto `update` può essere specificata la keyword `DEFAULT` per impostare un attributo sul valore di default del suo dominio

Esempio:

- Data la tabella `Impiegato(codice, nome, cognome, stipendio)`, la seguente query imposta il valore di default sull'attributo stipendio delle tuple rispettanti la condizione data

```
update Impiegato
    set stipendio = DEFAULT
    where codice = 1234
```

Observation 40. Sotto-query in inserimenti e cancellazioni

All'interno della clausola `where` di un costrutto `insert` o `delete` è possibile specificare anche una **query** rispettivamente per inserire o cancellare tutte le tuple generate dalla query

Esempio:

- La seguente query inserisce all'interno della tabella Persona tutte le tuple generate dalla query specificata

```
insert into Persona(nome, eta, stipendio)
    select padre, NULL, NULL from Paternita
    where padre not in (select nome from Persona)
```

- Analogamente, la seguente query elimina tutte le tuple generate dalla query specificata

```
delete from Paternita
    where figlio not in (select nome from Persona)
```

Observation 41. Espressioni e funzioni in update e insert

Come per la clausola `select`, anche all'interno delle clausole `update` e `insert` possono essere utilizzate **espressioni** e **funzioni**

Esempio:

- La seguente query inserisce una tupla nella tabella Esame utilizzando un'espressione matematica

```
insert into Esame(studente, corso, voto)
values ('1101', '593', 25+2)
```

4.5 Transazioni, Asserzioni e Trigger

Definition 43. Transazione

Una **transazione** è un insieme di operazioni svolte dal DBMS nel momento in cui viene invocata la transazione stessa.

Ogni transazione è:

- **Atomica e Indivisibile**: tutti gli effetti generati possono essere solo completamente confermati in blocco o completamente rifiutati in blocco, senza vie di mezzo, ripristinando lo stato iniziale del database nel caso in cui si verifichi un qualsiasi errore all'interno della transazione (**rollback**)
- **Consistente**: al suo termine, tutti i vincoli imposti sulla base di dati devono essere soddisfatti. In caso contrario, verrà effettuato un rollback allo stato iniziale
- **Isolata**: la sua esecuzione deve essere indipendente dalle altre transazioni

Per **avviare** una transazione, viene utilizzata la keyword `begin transaction`. Per **terminare e confermare** o per **abortire ed annullare** una transazione, vengono rispettivamente utilizzate le keyword `commit work` e `rollback work`

Esempio:

- Per effettuare un trasferimento sicuro di 1000€ dal conto corrente 111111 al conto corrente 999999, utilizziamo la seguente transazione

```
begin transaction;

update ContoCorrente
    set saldo = saldo - 1000
    where id = '111111'
update ContoCorrente
    set saldo = saldo + 1000
    where id = '999999'

commit work;
```

Observation 42. Vincoli deferred e non deferred

Un vincolo di integrità può essere dichiarato come **deferrable** (tramite la keyword omonima), ossia valutabile solo al termine di una transazione o valutabile anche durante una transazione al seguito di una qualsiasi operazione interna (a scelta dell'utente).

Di default, tutti i vincoli sono **non deferrable**, implicando che essi vengano valutati immediatamente dopo ogni operazione della transazione.

Definition 44. Asserzione

Un'**asserzione** è un costrutto SQL esprimente vincoli di integrità esterni ad una tabella ed arbitrariamente complessi

```
create assertion <nome_vincolo>
    check (<condizione>)
```

Esempio:

- Consideriamo il seguente database:
  - `Progetto(nome: varchar(100), inizio: date, fine: date)`

    Vincolo di tupla: `fine` $\geq$ `inizio`
  - `WorkPackage(nome: varchar(100), progetto: varchar(100), inizio: date, fine: date)`

    Vincolo di riferimento: `progetto references Progetto(nome)` Vincolo di tupla: `fine` $\geq$ `inizio`
- Vogliamo inoltre definire il seguente vincolo: inizio e fine di ogni WorkPackage devono essere compresi tra inizio e fine del corrispondente Progetto
- Per implementare tale vincolo, possiamo utilizzare i costrutti interni alla creazione della tabella WorkPackage:

```
create table WorkPackage (
    nome varchar(100) primary key,
    progetto varchar(100) not null,
    inizio date not null,
    fine date not null,
    foreign key progetto references Progetto(nome),
    check (fine >= inizio)
    check (inizio between
            date (select inizio from Progetto
                    where nome = progetto)
            and date (select fine from Progetto
                    where nome = progetto))
    check (fine between
            date (select inizio from Progetto
                    where nome = progetto)
            and date (select fine from Progetto
                    where nome = progetto))
)
```

- In alternativa, possiamo definire un'asserzione esterna:

```
create assertion check_date_wp check (
    not exists(
        select * from WorkPackage w, Progetto p,
        where w.progetto = p.nome
        and (w.inizio < p.inizio or w.fine > p.fine)
    )
)
```

Definition 45. Trigger

Un **trigger** è un costrutto invocante una procedura **prima**, **dopo** o **al posto di** (solo per le viste) un'**operazione intercettata** (la quale può essere `insert`, `update` o `delete`) su una tabella solo se la **condizione** imposta è verificata.

```
create [constraint] trigger <nome_trigger>
    {before | after | instead of}{<op_intercet> [or ...]} on
       <nome_tabella>
    [from <tabella_riferita>]
    { not deferrable | deferrable {
        initially immediate | initially deferred
    }}
    [for [each] {row | statement}]
    [when (<condizione>)]
    execute procedure <nome_procedura> (<argomenti>)
```

La keyword `for each row` indica che la procedura venga invocata una volta per ogni ennupla impattata dall'operazione, mentre la keyword `for statement` indica che la funzione venga invocata una volta per comando

Esempio:

- Data un'entity Persona, supponiamo che esista una disgiunzione non completa in due entity Studente e Docente. Per imporre tale disgiunzione all'interno del database, realizziamo una procedura in PL/pgSQL contenente il controllo da svolgere:

```
create function V_Persona_isa_disj()
    returns trigger as $V_Persona_isa_disj$

    declare isError boolean := false;
    begin
        case --TG_TABLE_NAME: nome della tabella relativa all'op
            when TG_TABLE_NAME ilike 'Studente' then
                isError = exists (
                    select * from Docente d
                    where d.persona = new.persona
                    --new e' la tupla da aggiungere
                );
            when TG_TABLE_NAME ilike 'Docente' then
                isError = exists (
                    select * from Studente s
                    where s.persona = new.persona
                );
            else raise exception
                "La procedura non puo' essere invocata sulla
                   tabella %", TG_TABLE_NAME;
        end case;
        if (isError) then raise exception
            "Vincolo V.SDisaP.disjoint violato da Persona.id =
               %", new.persona;
        end if;
        return new;
    end;
    $V_Persona_isa_disj$ language plpgsql;
```

- Successivamente, creiamo due trigger che vadano ad utilizzare tale procedura:

```
create trigger V_Persona_isa_disj_trigger_Studente
before insert or update on Studente
for each row execute procedure V_Persona_isa_disj();


create trigger V_Persona_isa_disj_trigger_Docente
before insert or update on Docente
for each row execute procedure V_Persona_isa_disj();
```