

Basi Di Dati

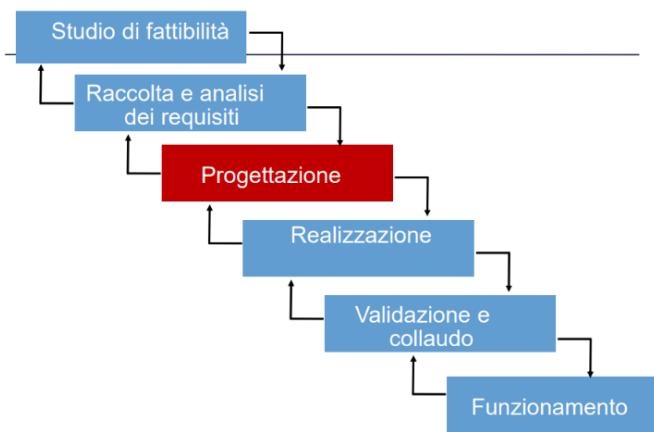
Anno 2022/2023

A CURA DI:
Erica Corda
Matteo Dessì
Leonardo Dessì
Simone Giuffrida

MODELLO ENTITA-RELAZIONE E PROGETTAZIONE CONCETTUALE

In un sistema informativo complesso (ad es. di un'azienda o di un'università) il database sarà costituito da molte tabelle, le quali saranno collegate fra di loro mediante delle relazioni, che noi chiameremo associazioni per evitare la confusione di relazione intesa come tabella. Per questo motivo abbiamo la necessità di sapere da dove cominciare per definire le tabelle, e come suddividere le informazioni in tabelle, serve quindi una **metodologia di progettazione**.

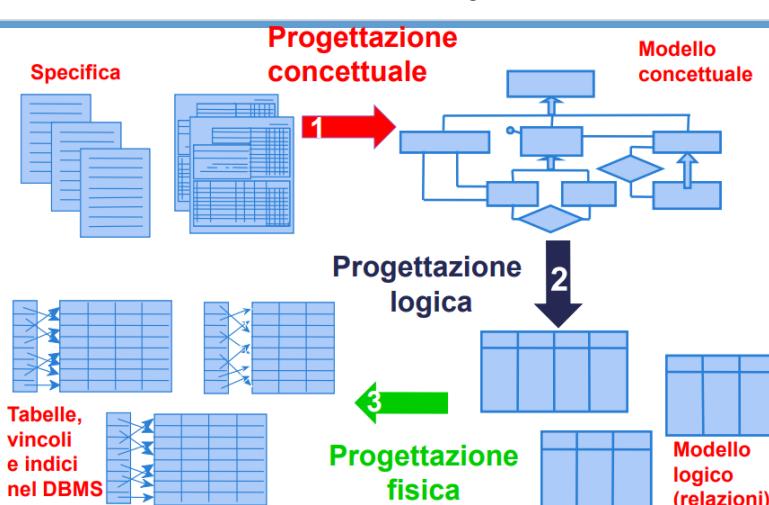
La **progettazione** di una base di dati è una delle attività del processo di sviluppo di un sistema informativo, e deve tenere conto del **ciclo di vita** di quest'ultimo, il quale è un insieme delle attività svolte da analisti, progettisti, utenti, nello sviluppo e nell'uso dei sistemi informativi. Viene chiamato **ciclo** perché è **un'azione iterativa** perché ad esempio le azioni di testing e di utilizzo vengono ripetute più volte.



Le fasi principali sono queste: si parte dallo **studio di fattibilità** con cui si fa un'analisi costo benefici e si capisce se ha senso creare quell'applicazione. Se ha senso si passa alla **raccolta e analisi dei requisiti**, dopodiché si passa alla **progettazione vera e propria** del sistema che coinvolgerà diversi aspetti, come l'aspetto software con cui si decide quale sistema operativo utilizzare, quale linguaggio utilizzare, ecc., altri aspetti sono quale tipo di DBMS utilizzare, quali tabelle bisogna definire.

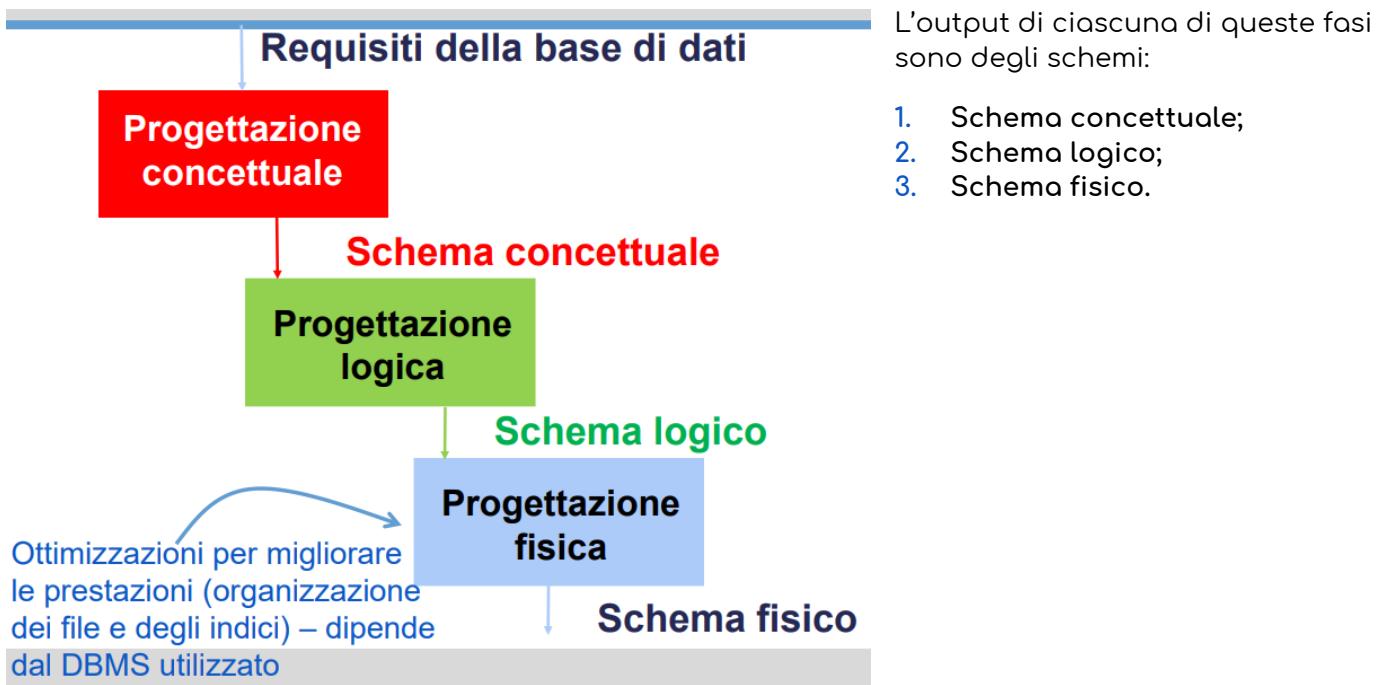
Poi si passa alla **realizzazione** quindi implementare il software e il database, poi c'è la fase di **validazione e collaudo** con cui si effettuano dei testing con cui si controlla se il sistema è effettivamente funzionante, e lo si fa provare anche agli utenti. Dopo tutte le fasi il sistema va in **produzione**, quindi, viene messo nel mercato.

Come notiamo nello schema in ogni fase c'è una freccia che porta alla fase successiva, ma anche un freccia che porta a quella precedente, questo perché se magari nella fase di testing ci si rende conto che manca una funzionalità bisogna tornare indietro e implementarla.



La **progettazione concettuale** parte da una **specifiche** fornita da colui che vuole un determinato sistema, la fase successiva è la **realizzazione del modello concettuale** che è un modello costituito da uno **schema grafico** che rappresenta le entità e le diverse frecce rappresentano le relazioni fra le diverse entità.

Il passo successivo è la **progettazione logica** che prevede la "traduzione" del modello concettuale nel modello logico, con cui vengono definite le tabelle, infine si passa alla **progettazione fisica** che riguarda l'implementazione delle tabelle nel database, con la definizione dei vincoli e degli indici.



Come abbiamo detto prima il punto di partenza è l'analisi dei requisiti, le cui fonti possono essere di diverso tipo:

- Forniti da utenti e committenti, attraverso interviste, documentazione apposita;
- Mediante della documentazione esistente: come le normative (leggi, regolamenti di settore), i regolamenti interni, le procedure aziendali e realizzazioni preesistenti.

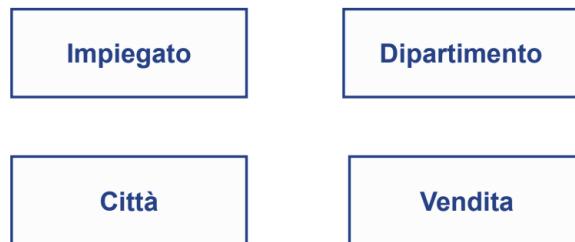
La **progettazione concettuale** viene realizzata attraverso il modello **Entità-Relazione** (ER), il quale è il modello più diffuso. È una rappresentazione prevalentemente grafica e si basa su:

- Entità;
- Associazione ("Relationship");
- Attributo;
- Chiave;
- ...altri aspetti.

Un'**entità** è un classe di oggetti (fatti, persone, cose) della realtà di interesse con proprietà (attributi) comuni e con esistenza "autonoma". Ogni entità diventerà una tabella all'interno del database.

Quindi un' entità è una classe di oggetti, persone, .., "omogenei", un' **occorrenza** (o istanza) di entità è un elemento della classe (l'oggetto, la persona). Nello schema concettuale rappresentiamo le entità, non le singole istanze.

Le entità vengono rappresentate con dei **quadrati**:



Ogni **entità** ha un nome che la identifica univocamente nello schema, un'opportuna convenzione è quella di utilizzare il **nome al singolare**, e deve essere un **nome espressivo** che faccia capire immediatamente cosa vogliamo descrivere.

Un'associazione è un legame logico fra due o più entità, rilevante nell'applicazione di interesse. Ad esempio:

- Residenza (fra persona e città);
- Esame (fra studente e corso).

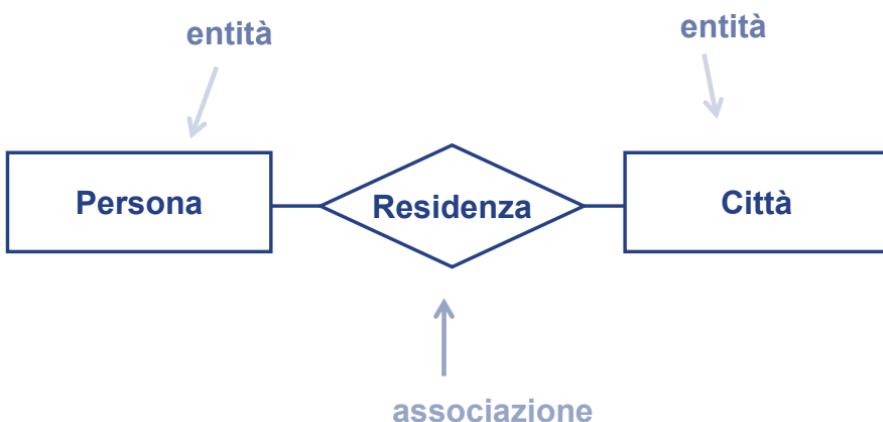
A volte le associazioni possono diventare nuove tabelle, ma non sempre.

Le associazioni vengono identificate con dei rombi:

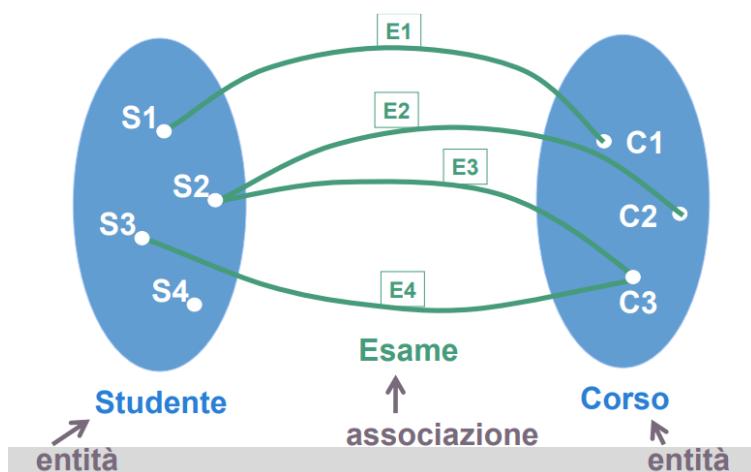


Ogni associazione ha un nome che la identifica univocamente nello schema, la regola generale indica che bisogna utilizzare un nome espressivo, al singolare, e preferire l'utilizzo di sostantivi rispetto ai verbi.

Uno schema ER è così composto quindi:



Esempi di occorrenze



ESERCIZIO

Considerare le informazioni per la gestione dei prestiti di una biblioteca personale. Il proprietario presta libri ai suoi amici. Di ogni amico conosce il nome, numero di telefono e indirizzo. Di ogni libro tiene traccia del nome e del prezzo. Per ogni libro in prestito, prende nota della data del prestito e di quella prevista di restituzione.

Definire entità e associazioni dello schema ER per il database descritto sopra:



Una occorrenza di una associazione binaria è **coppia di occorrenze di entità**, una per ciascuna entità coinvolta. Una occorrenza di una associazione n-aria è una **n-upla** di occorrenze di entità, una per ciascuna entità coinvolta.

Nell'ambito di una associazione non ci possono essere occorrenze (coppie, tuple) ripetute, quindi in ER, se due entità sono legate da una relazione, non possono avere coppie di istanze ripetute.



Quindi, ad esempio, un cliente non può comprare 2 volte lo stesso prodotto o una persona non può risiedere 2 volte nella stessa città



VA BENE QUESTO MODELLO?



In caso non andasse bene, perché magari vogliamo tenere per gli esami anche le insufficienze, che non verrebbero inserite in Corso, la soluzione è quella di **trasformare le associazioni in entità**. Quest'operazione viene chiamata **Reificazione** o **Materializzazione**.

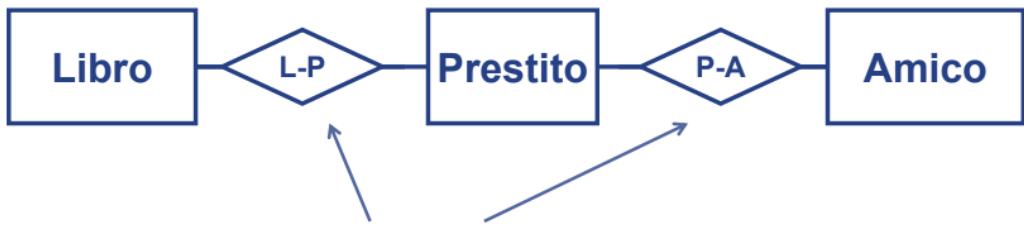
ESERCIZIO

➤ Considerando l'esercizio precedente, va bene questa modellazione?



In caso negativo, come la devo modificare?

In generale, nulla vieta di prestare più volte lo stesso libro allo stesso amico (in date diverse). Quindi "Prestito" non può essere una associazione.



In questo caso non trovo termini adatti per le associazioni; uso delle sigle

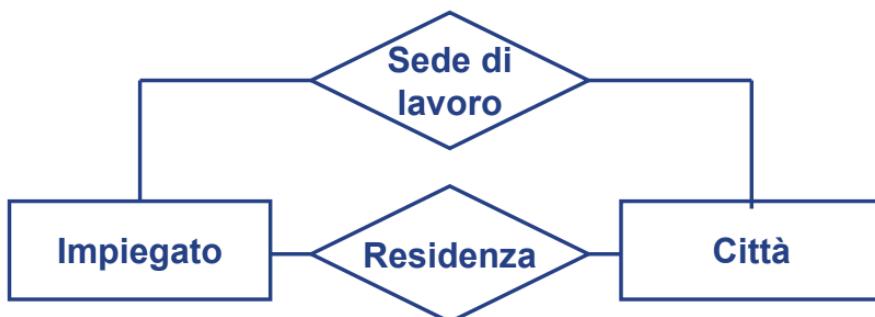
- Considerare le informazioni per la gestione delle presenze di pesci nelle vasche di un acquario civico. Ogni tipo di pesce ha un nome univoco (es. Piranha), e se ne conosce la famiglia (ciclidi, caracidi, ecc). Ogni vasca ha un nome univoco, e se ne conosce la sala, il piano e la capienza in litri.

Si vuole tenere traccia delle presenze e della numerosità dei pesci nelle vasche.
Definire entità e associazioni dello schema ER per il database descritto sopra:



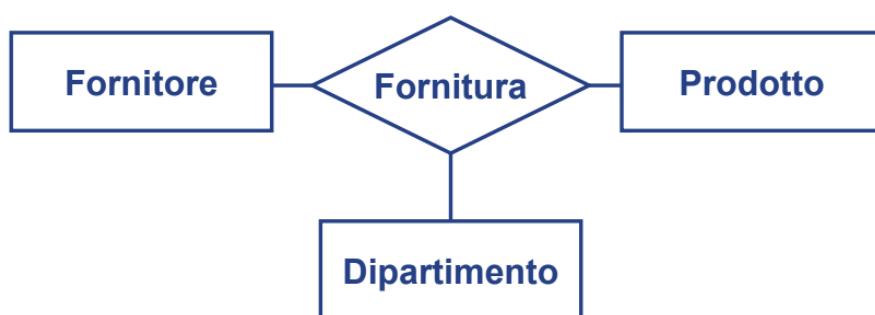
Va bene così o devo materializzare "Presenza"? Va bene così, perché una volta che conosco che c'è un determinato pesce nelle vasca non mi interessa avere questo valore ripetuto. Se invece volessi sapere i pesci presenti ogni giorno, quindi tenere uno storico, bisognerebbe materializzare presenza.

Può capitare anche di [avere due associazioni sulle stesse entità](#):



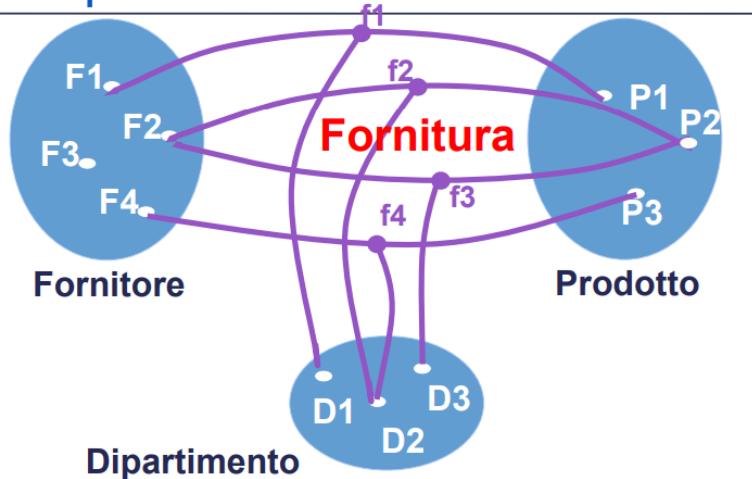
In questo caso la stessa persona può risiedere in una città e avere sede di lavoro diversa.

Può anche esistere [un'associazione n-aria](#), cioè un'associazione non binaria:



Degli esempi di occorrenze di questo sistema sono:

Esempi di occorrenze



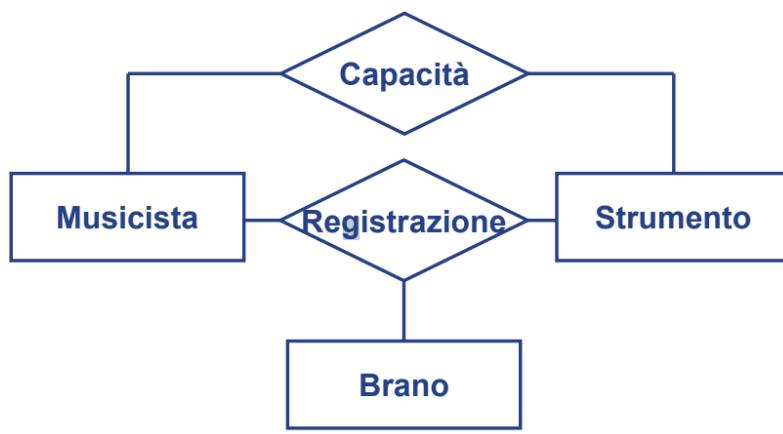
Quindi avremo sostanzialmente una tripla di valori.

Esercizio:

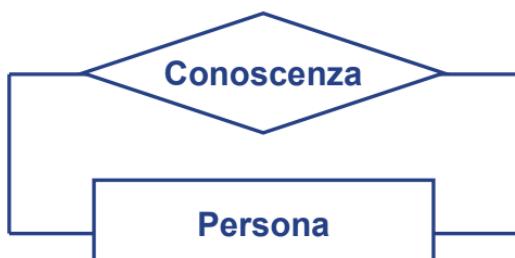
Considerare le informazioni per la gestione di musicisti, strumenti e brani. Di ogni musicista si conosce nome, cognome, e strumenti che sa suonare. Degli strumenti si conosce il nome e il tipo (a corda, a fiato, ecc). Dei brani si conosce il titolo, l'anno, e chi ha suonato quale strumento durante la sua registrazione.

Definire entità e associazioni dello schema ER per il database descritto sopra:

Come **entità** abbiamo quindi il musicista, lo strumento e il brano. Noi vogliamo sapere chi ha suonato quale strumento durante la registrazione del brano; quindi, avremo sicuramente un'**associazione registrazione** che collega il musicista con il brano e con lo strumento, inoltre il musicista può registrare brani con un determinato strumento, ma può essere capace anche di suonare più strumenti, per questo motivo creiamo anche un'**associazione capacità** fra musicista e strumento.



Un altro particolare tipo di **associazione** è quella **ricorsiva** che coinvolge due volte o più la stessa **entità**:



In questo caso conoscenza è un' associazione che mette in relazione due entità Persona, in questo caso è anche [simmetrica](#), ma a volte accade che non sia simmetrica.

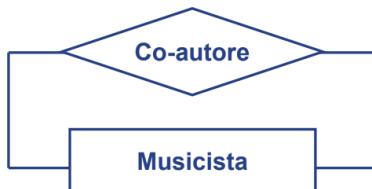


In questo caso la ricorsione **non è simmetrica**, in quanto da un lato troviamo i successori e dall'altro i predecessori, per indicare questi è necessario utilizzare un'[etichetta](#). Senza l'utilizzo di etichette si fa per scontato che l'associazione sia simmetrica.

ESERCIZI

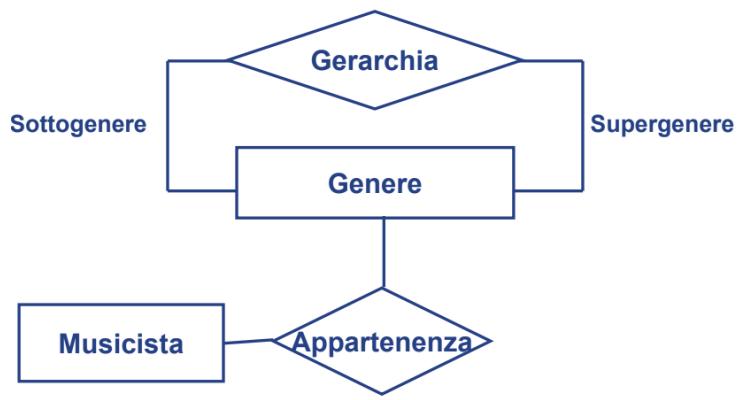
- Considerare le informazioni per la gestione di musicisti. Di ogni **musicista** si conosce nome, cognome, e con quali altri musicisti ha scritto dei brani insieme.

Definire entità e associazioni dello schema ER per il database descritto sopra:



- Considerare le informazioni per la gestione di musicisti e generi musicali. Di ogni **musicista** si conosce nome, cognome e nazionalità, e i generi di musica che suona. Di ogni **genere musicale** si conosce il nome, l'epoca, e i suoi sottogeneri.

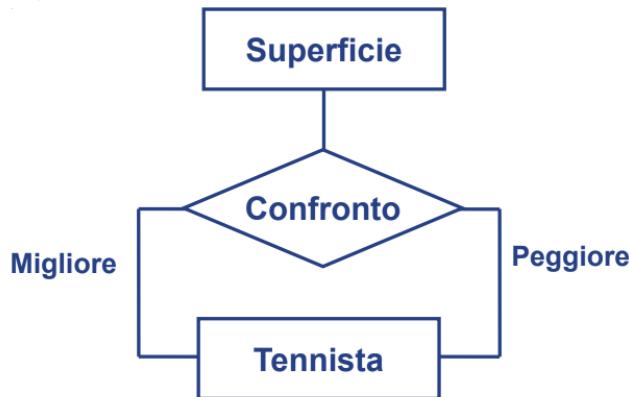
Definire entità e associazioni dello schema ER per il database descritto sopra:



Partiamo dall'entità **musicista**, e sappiamo a quale genere appartiene il tipo di musica del musicista; quindi, creiamo un'associazione "[appartenenza](#)" fra musicista e genere. Dopodiché di ogni genere si conosce il sottogenere, che è anch'esso un genere quindi creiamo un'associazione ricorsiva "[gerarchia](#)" con cui specifichiamo, tramite l'utilizzo delle etichette, se il genere è un sottogenere o supergenere di un altro genere.

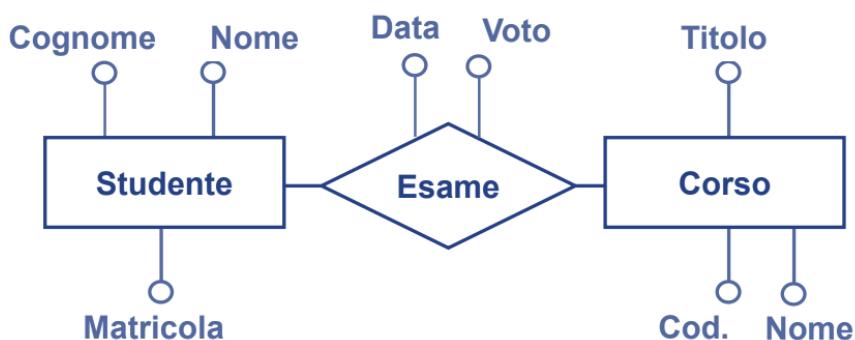
- Considerare le informazioni per la gestione di tennisti e tipi di superficie (terra battuta, cemento, ecc). Di ogni **tennista** si conosce **nome**, **cognome** e **nazionalità**. Di ogni **superficie** si conosce il **nome** e **l'anno di introduzione**. Per alcune coppie di tennisti, si sa anche chi è il migliore tra i due su una certa superficie.

Definire entità e associazioni dello schema ER per il database descritto sopra:



Partiamo dalla creazione delle entità **tennista** e **superficie**, per determinare quale tennista è il migliore/peggiore in una determinata superficie creiamo un'associazione ricorsiva **confronto** collegata al tennista, con la quale, tramite l'utilizzo delle etichette, individuiamo il tennista migliore e peggiore. Per capire in quale superficie il tennista è il migliore/peggiore colleghiamo con un arco da superficie verso confronto.

Un **attributo** è una proprietà elementare di un'entità o di una associazione, di interesse ai fini dell'applicazione. Associa ad ogni istanza di entità o associazione un valore appartenente a un insieme detto **dominio dell'attributo** (che ne indica il tipo).

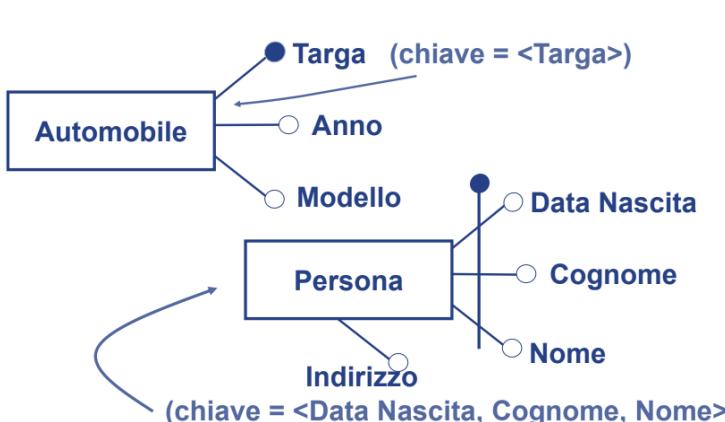


Un **identificatore di un'entità o chiave** è uno strumento che viene utilizzato per identificare in maniera univoca delle istanze di un'entità. È costituito da:

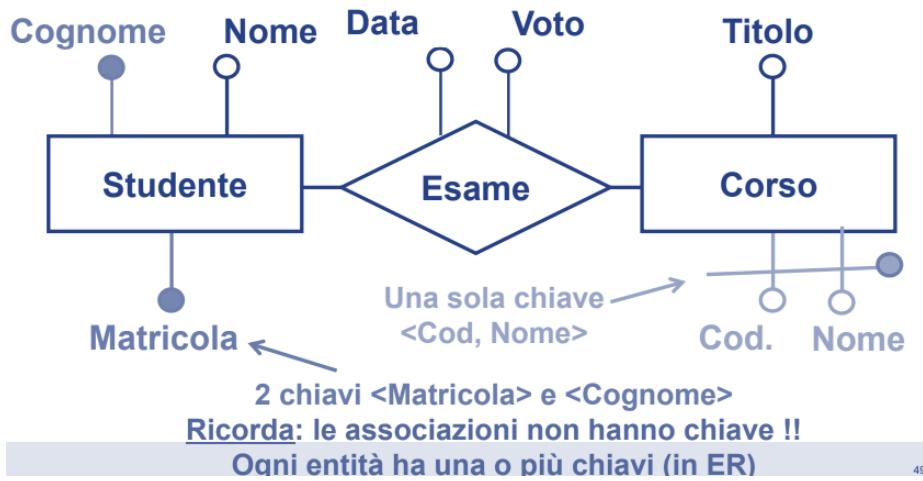
- Un'**identificazione interna**: attributi dell'entità;
- e/o un'**identificazione esterna**: tramite gli attributi di un'altra entità [direttamente collegata tramite un'associazione](#).

Ogni entità deve possedere almeno una chiave, ma può averne più di una, nella progettazione logica si sceglierà qual è la chiave primaria tra tutte le chiavi dell'entità.

Non **esistono chiavi nelle associazioni**:



In questo esempio abbiamo l'entità **Automobile** che possiede tre attributi **targa**, **anno** e **modello**, in cui targa rappresenta la chiave. E abbiamo un'altra entità **Automobile** con attributi **data nascita**, **cognome**, **nome** e **indirizzo** e la chiave è composta dall'insieme di attributi **data nascita**, **cognome** e **nome**.



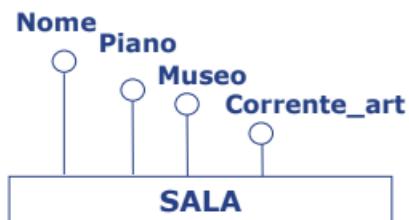
In questo caso abbiamo le entità Studente e Corso con chiavi rispettivamente cognome e matricola (di cui se ne dovrà decidere solo una), e nome e codice (l'unione di essi).

Notiamo che fra queste entità vi è un associazione esame con attributi data e voto, ma nessuna chiave, questo perché come abbiamo detto prima le associazioni non hanno chiave.

ESERCIZI

- ↗ Data la seguente entità:

Stabilire il suo identificatore nelle diverse ipotesi:

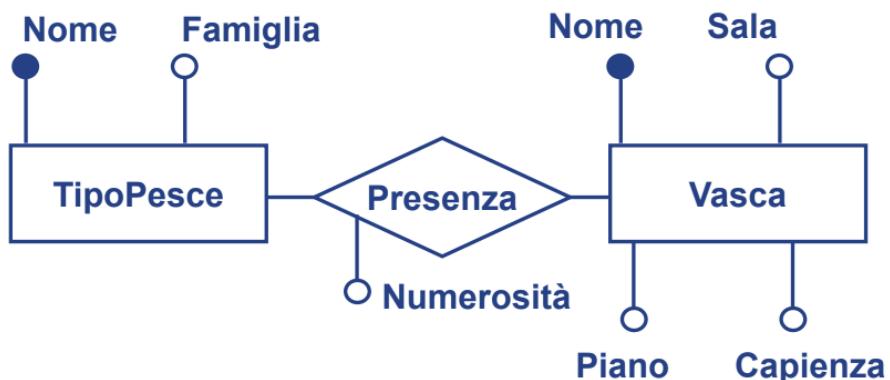


1. Nei musei considerati non vi sono sale con nomi uguali.
2. Nei musei considerati su uno stesso piano non vi sono sale con lo stesso nome.
3. Nei musei considerati le sale che espongono opere di una stessa corrente artistica hanno nomi diversi.
4. In ciascun museo, su ogni piano le sale hanno nomi diversi.

1. Nel primo caso possiamo decidere come chiave il nome, in quanto non esistono sale con nome uguale;
2. Nel secondo caso come chiave possiamo decidere la coppia <nome, piano>;
3. In questo caso la chiave sarà costituita dalla coppia <nome, corrente artistica>;
4. In questo caso la chiave sarà costituita dalla tripla <nome, piano , museo>.

- ↗ Considerare le informazioni per la gestione delle presenze di pesci nelle vasche di un acquario civico. Ogni **tipo di pesce** ha un **nome** univoco (es. Piranha), e se ne conosce la **famiglia** (ciclidi, caracidi, ecc). Ogni **vasca** ha un **nome** univoco, e se ne conosce la **sala**, il **piano** e la **capienza** in litri. Si vuole tenere traccia delle presenze e della numerosità dei pesci nelle vasche.

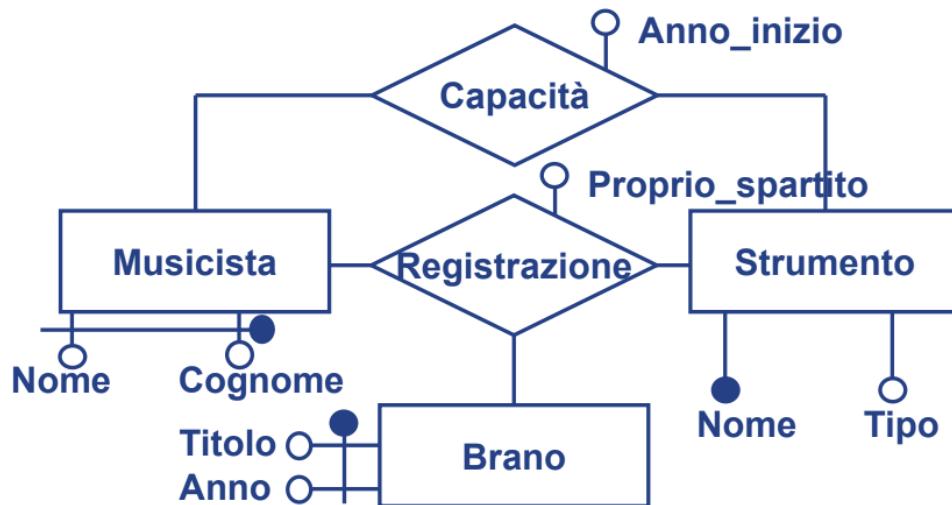
Aggiungi attributi e chiavi allo schema seguente:



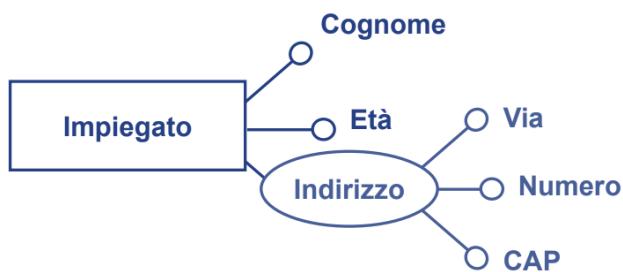
- ↗ Considerare le informazioni per la gestione di musicisti, strumenti e brani. Di ogni **musicista** si conosce **nome**, **cognome**, e **strumenti** che sa suonare. Per ogni **strumento** che sa suonare si

conosce l'anno in cui l'ha imparato. Degli **strumenti** si conosce il **nome** e il **tipo** (a corda, a fiato, ecc). Dei **brani** si conosce il **titolo**, l'**anno**, e chi ha suonato quale strumento durante la sua registrazione. Si sa anche se ha suonato quello strumento su un **proprio spartito** o su **spartito altrui**.

Aggiungi attributi e chiavi allo schema seguente:

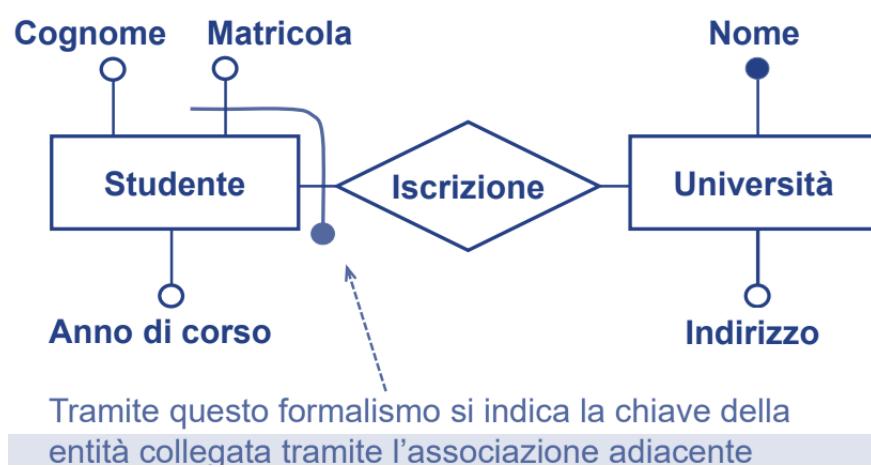


Gli **attributi composti** raggruppano attributi di una medesima entità o associazione che presentano affinità nel loro significato o uso, come ad esempio: via, numero civico e CAP formano un indirizzo.



Quando la chiave è composta da più attributi la notazione è: **<Titolo, Anno>** questo però non significa che sia un attributo composto.

Le **chiavi esterne** permettono di identificare un'entità (in tutto o in parte) utilizzando gli identificatori di un'altra entità a cui è collegata direttamente tramite un'associazione.



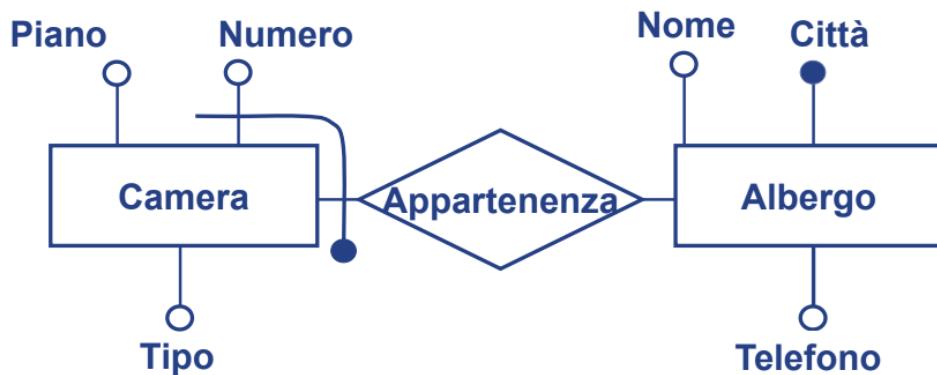
In questo esempio lo studente è identificato sia dalla sua Matricola che dalla chiave di Università (Nome).

Quindi la chiave di studente sarà la coppia **<Matricola, Nome>**, dove Nome quindi sarà la chiave esterna.

ESERCIZI

- Considerare le informazioni per la gestione delle camere di una catena di alberghi. Di ogni **albergo** si conosce **nome**, **città** e **telefono**. Esiste al più un albergo della catena in ogni città. Di ogni **camera** si conosce il **tipo** (singola, matrimoniale), il **numero**, il **piano**, e l'**albergo** in cui si trova.

Definire lo schema ER per il database descritto sopra, indicando entità, associazioni, attributi e chiavi:



Creiamo l'entità **albergo** con gli attributi nome, città e telefono, scegliamo come chiave la città in quanto ci può essere al più un albergo della catena in ogni città, creiamo poi l'entità **camera** con gli attributi tipo, numero e piano e la colleghiamo con l'entità albergo con l'associazione appartenenza.

La chiave primaria di Camera è la coppia **<Numero, Città>** questo perché ad alberghi presenti in città diverse può corrispondere lo stesso numero di stanza, e grazie alla coppia riusciamo ad identificare univocamente una camera.

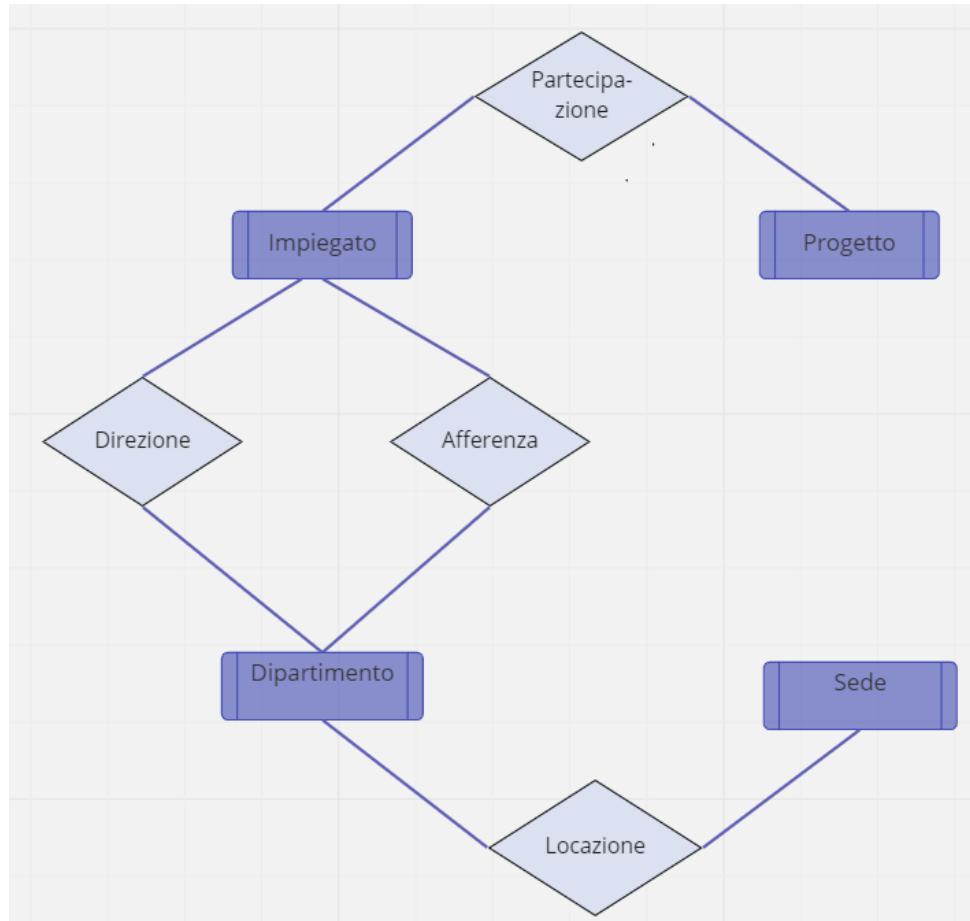
- Considerare le informazioni per la gestione di una azienda. Di ogni **impiegato** si conosce il **codice** e il **cognome**, a quali **dipartimenti** afferisce, e da quale **anno** ha cominciato ad afferirvi. Del **dipartimento** si conosce il **nome**, il **numero di telefono**, il **direttore**, e la **sede**. Possono esistere dipartimenti con lo stesso nome purchè in sedi diverse. Di ogni **sede** si conosce l'**indirizzo** (via, cap) e la **città**. Gli impiegati partecipano a **progetti** di cui si conosce il **nome** e il **budget**.

Disegna lo schema ER (ent., assoc., attrib. e chiavi):

Per prima cosa troviamo le entità, che in questo caso sono Impiegato, Dipartimento, Sede e Progetto.



Dopodiché [troviamo le associazioni](#), nelle specifiche vediamo che ogni impiegato afferisce ad un dipartimento; quindi, creiamo un'associazione fra impiegato e dipartimento ([Afferenza](#)). Di ogni dipartimento conosciamo il direttore, quindi questa sarà un'altra associazione fra Impiegato e Dipartimento ([Direzione](#)). Del Dipartimento conosciamo anche la Sede, quindi creiamo un'associazione fra queste entità ([Locazione](#)). Infine, ogni Impiegato lavora ad un Progetto, quindi creiamo un'ulteriore associazione ([Partecipazione](#)).

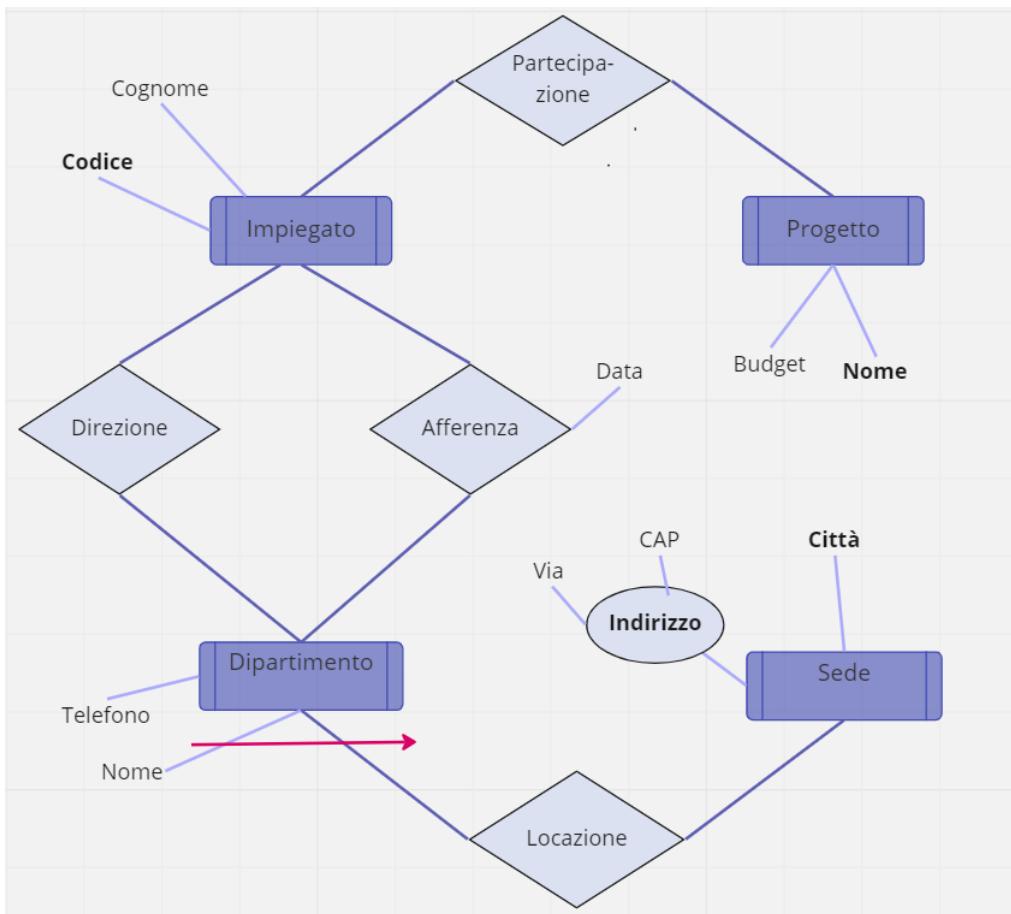


Adesso dobbiamo andare ad [inserire gli attributi](#):

- ↗ Per ogni **Impiegato** sappiamo [codice](#) e [cognome](#);
- ↗ Di ogni **impiegato** sappiamo anche la [data](#) in cui afferisce in un dipartimento; quindi, inseriamo l'attributo [data](#) nell'associazione **Afferenza**;
- ↗ Di ogni **Dipartimento** conosciamo il [nome](#) e il [telefono](#), conosciamo anche il direttore ma questa l'informazione la estrapoliamo dall'associazione **Direzione**;
- ↗ Di ogni **Sede** conosciamo l'[indirizzo](#) che è un attributo composto da [via](#) e [CAP](#), e la città;
- ↗ Di ogni **Progetto** conosciamo il [budget](#) e il [nome](#);

L'ultima cosa che dobbiamo fare è stabilire le chiavi primarie. Abbiamo:

- Per **Impiegato**, il [codice](#);
- Per **Dipartimento**, l'insieme delle chiavi [`<nome, sede>`](#);
- Per **Sede** l'insieme delle chiavi [`<indirizzo, città>`](#), in modo tale che ci possano essere più sedi nella stessa città, ma in indirizzi diversi.
- Per **Progetto**, il [nome](#).



Un'altra cosa molto importante nella progettazione concettuale è quella di [determinare la cardinalità delle associazioni](#). Quest'ultima specifica il numero minimo e massimo di occorrenze delle associazioni cui ciascuna occorrenza di una entità può partecipare:



In questo esempio un impiegato è assegnato ad un determinato incarico, dobbiamo stabilire le cardinalità delle associazioni. Il numero **(2,5)** ci indica che un impiegato può essere assegnato ad un minimo di 2 incarichi fino ad un massimo di 5, invece l'incarico può essere assegnato ad un minimo di 0 (nessuno) impiegati fino ad un massimo di 50 impiegati.

Soltanmente questo tipo di rappresentazione specifica non viene utilizzata, ma si utilizza una [rappresentazione più semplificata](#), che utilizza tre simboli:

- **0** e **1** per la cardinalità minima:
 - 0 = "partecipazione opzionale" (un incarico può anche essere assegnato a nessuno);
 - 1 = "partecipazione obbligatoria" (deve partecipare almeno un istanza di un'entità, quindi nell'esempio di prima un incarico deve essere assegnato almeno a qualcuno);
- **1** e "**N**" per la massima: dove N non pone alcun limite, se metto 1 un impiegato non può avere più di un incarico.

Le possibili combinazioni sono: **(0,1)** **(0,N)** **(1,1)** **(1,N)**.



In questo esempio un impiegato può essere assegnato a più di un incarico, ma un incarico può essere assegnato ad uno e un solo impiegato.

ESERCIZI:



In questo esempio uno studente può risiedere in un'unica città, ma in una città possono risiedere sia 0 studenti che un numero illimitato di studenti.



In questo esempio invece, uno studente può aver dato 0 esami in un corso (quando si è appena scritto) fino a un numero N di esami, invece in un corso ci possono essere sempre 0 esami dati (magari è un nuovo corso, e nessuno ha ancora sostenuto esami) fino ad un numero N.



Qui possiamo avere montagne che non sono mai state scalate da nessuno quindi 0, oppure da N alpinisti, un'alpinista, essendo tale, deve aver almeno scalato una montagna fino ad un massimo di N.



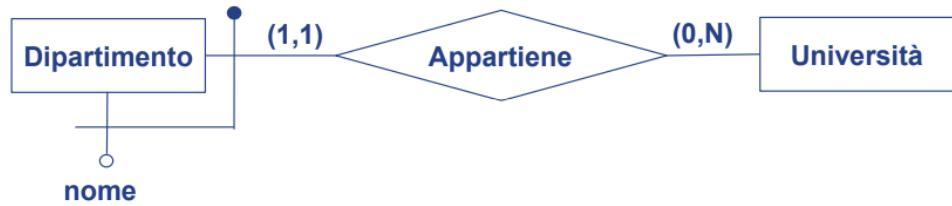
Ogni docente per essere tale deve almeno tenere un corso, ma ne può tenere anche più di uno, ma uno corso deve essere sostenuto da esattamente un docente.

ESERCIZIO:

Spiega il significato di questi schemi e evidenzia le differenze:



Il primo ministro per essere tale deve governare per forza una nazione, e ne può governare una sola, una nazione può in un dato momento non aver alcun ministro, perché magari si è dimesso, e al massimo ne può avere uno solo. Quella freccia vicino a Primo Ministro indica che esso è identificato dalla nazione che governa, che rappresenta sia la sua chiave esterna che chiave primaria.



Il dipartimento può appartenere ad un' unica università, ma all'università appartengono da un minimo di un dipartimento fino ad un massimo di N (ha detto che l'esempio è sbagliato, ed è meglio mettere 1 al posto di 0). In questo caso il Dipartimento è identificato dal proprio nome e dall'Università.

ESERCIZIO

Dato il seguente schema ER:



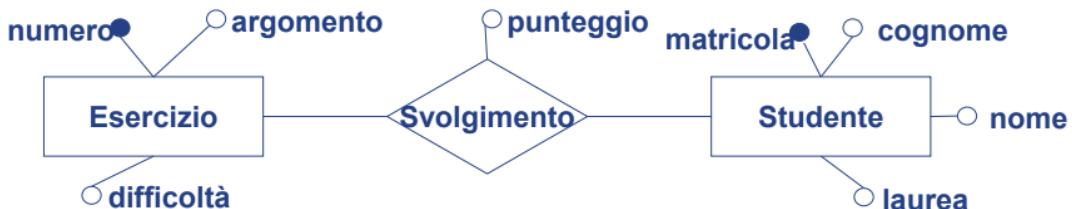
Stabilire quali delle seguenti affermazioni sono vere:

1. Un disco è inciso da un solo cantante;
2. Un artista non può incidere più di N dischi;
3. 2 artisti con stesso id devono avere nomi diversi;
4. Ci sono artisti che non hanno inciso dischi;
5. Si possono avere 2 dischi con lo stesso titolo e lo stesso codice.

1. Falsa, perché se fosse così la cardinalità massima di disco sarebbe 1;
2. Falsa, N rappresenta un valore illimitato;
3. Falsa, non possono esistere due artisti con id uguali essendo chiave;
4. Falsa, il minimo di cardinalità di Artista è 1 non 0;
5. Falsa, codice è chiave quindi non posso avere due dischi con stesso codice.

Non c'è né una vera, Riboni si diverte a fare le cose a trabocchetto.

ESERCIZIO



Determinare le cardinalità della relazione nei seguenti casi:

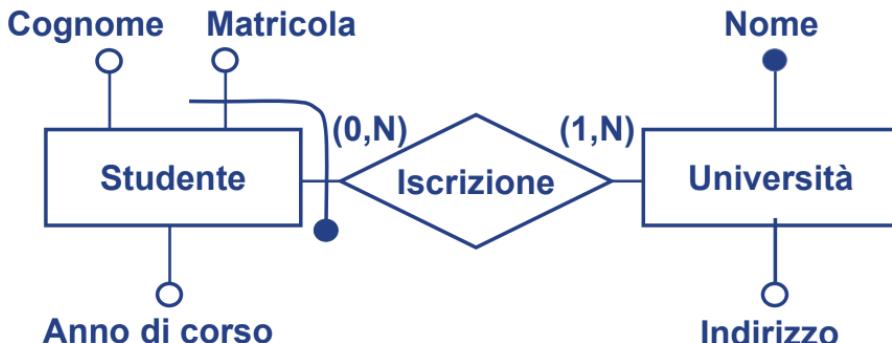
1. Un esercizio è svolto da almeno uno studente. Ogni studente svolge un solo esercizio;
2. Ci sono esercizi non svolti e altri svolti da più studenti. Alcuni studenti non hanno svolto esercizi, altri ne hanno svolti molti;
3. Ogni esercizio è stato svolto da almeno uno studente ma anche da più studenti. Ogni studente deve svolgere almeno un esercizio.

1. Il minimo di cardinalità da esercizio è 1, e il massimo è N, la cardinalità di studente invece è 1,1;

- Il minimo di cardinalità da esercizio è 0, e il massimo è N, la cardinalità di studente è 0,N;
- Il minimo di cardinalità da esercizio è 1, e il massimo è N, la cardinalità di studente è 1,N.

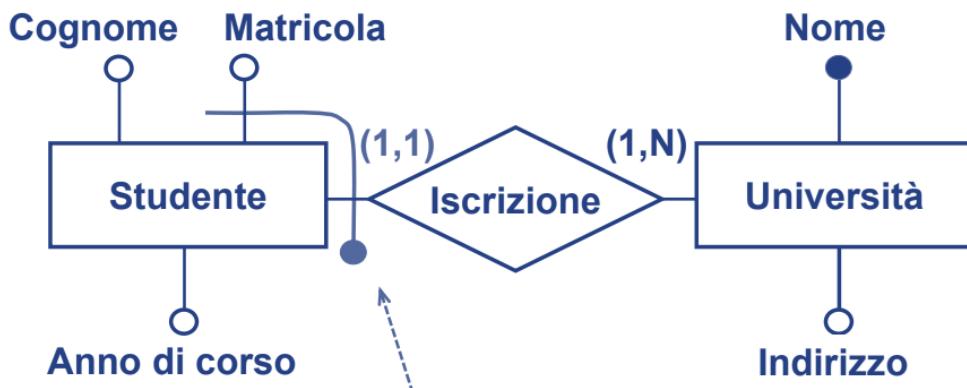
Esercizio

Trova l'errore:



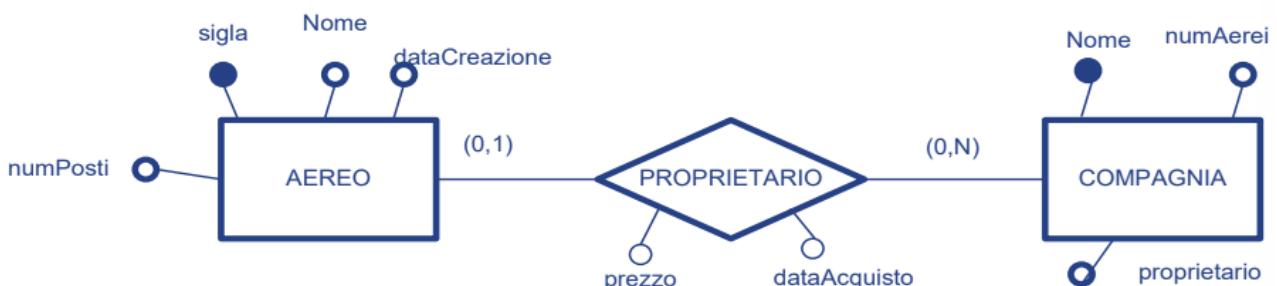
Con questa notazione uno studente è identificato dalla sua matricola e dal nome dell'università, ma la cardinalità associata è (0,N) quindi lo studente potrebbe essere iscritto anche a nessuna università, ma in questo modo non lo potremmo identificare non avendo la chiave esterna.

E la cardinalità massima dovrebbe essere 1, perché avendo come chiave anche la matricola, avremo la stessa matricola associata a più di un'università; quindi, non potrei sapere a quale università si riferisce la matricola.



L'entità è identificata (in tutto o in parte) usando gli identificatori di un'altra entità a cui è collegata direttamente tramite un'associazione. Una **identificazione esterna** è possibile solo attraverso una associazione a cui l'entità da identificare partecipa con cardinalità (1,1).

Esercizio



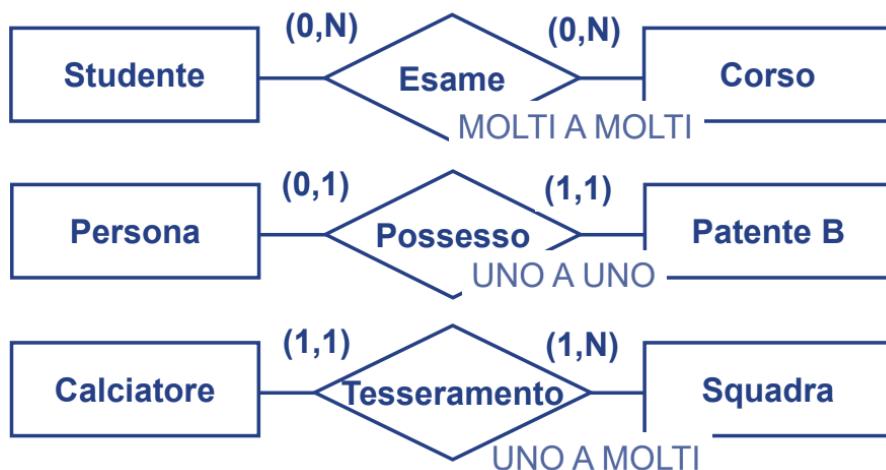
Si consideri lo schema ER e si indichi quale delle seguenti affermazioni è vera:

- Ci possono essere più aerei della stessa compagnia;
- Ci possono essere più compagnie con lo stesso nome;
- Ci sono aeroplani che non appartengono ad alcuna compagnia;
- Ogni compagnia possiede almeno un aereo;
- Un aereo può appartenere a compagnie diverse.

1. Vera, una compagnia può avere da un minimo di 0 ad un massimo di N aerei;
2. Falsa, la chiave di compagnia è nome, quindi non possiamo più compagnie con lo stesso nome;
3. Vera, la cardinalità minima di Aereo è 0;
4. Falsa, la cardinalità minima di Compagnia è 0;
5. Falsa, perché un aereo appartiene al massimo ad una compagnia.

Con riferimento alle [cardinalità massime](#), abbiamo associazioni:

- Uno a uno;
- Uno a molti;
- Molti a molti.



Uno studente può dare più esame, un esame può essere sostenuto da più studenti.

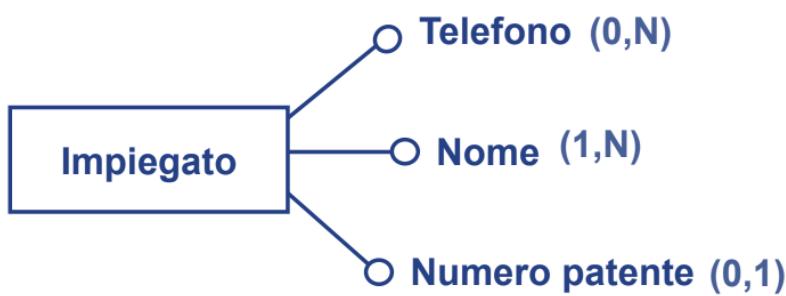
Una persona può essere in possesso di una sola patente B, la patente può essere posseduta da una singola persona (si intende la singola istanza di quella patente).

Un calciatore può essere tesserato in un'unica squadra, in una squadra possono essere tesserati più calciatori.

Questo ci sarà utile quando trasformeremo le nostre entità in tabelle.

È possibile [associare delle cardinalità anche agli attributi](#), con due scopi:

1. indicare opzionalità ("informazione incompleta");
2. indicare attributi multivaleore.



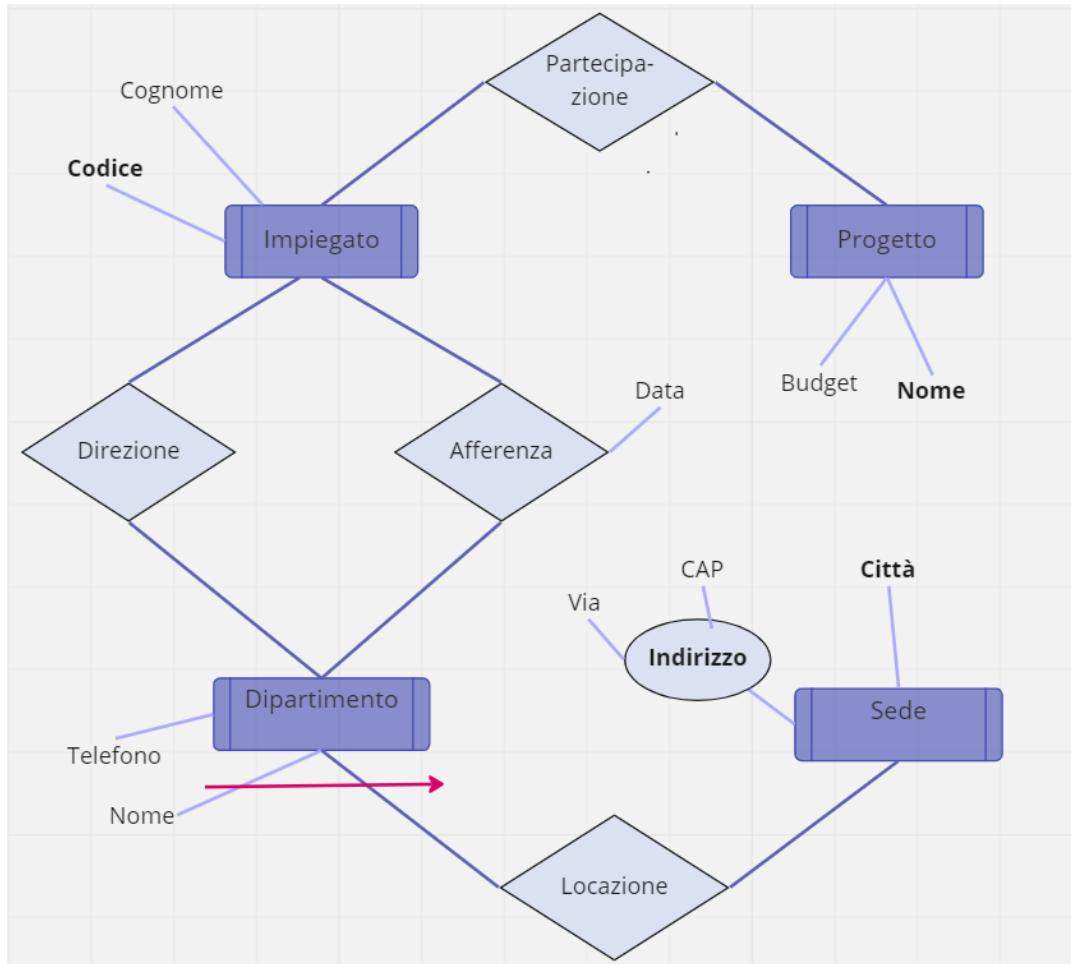
Un impiegato può avere 0 numeri di telefono o più di uno, e così via. Se non specificassimo nulla [quello di default è \(1,1\)](#).

Gli attributi con cardinalità opzionale o multipla non possono essere chiave (non garantiscono che possa identificare le istanze della relazione).

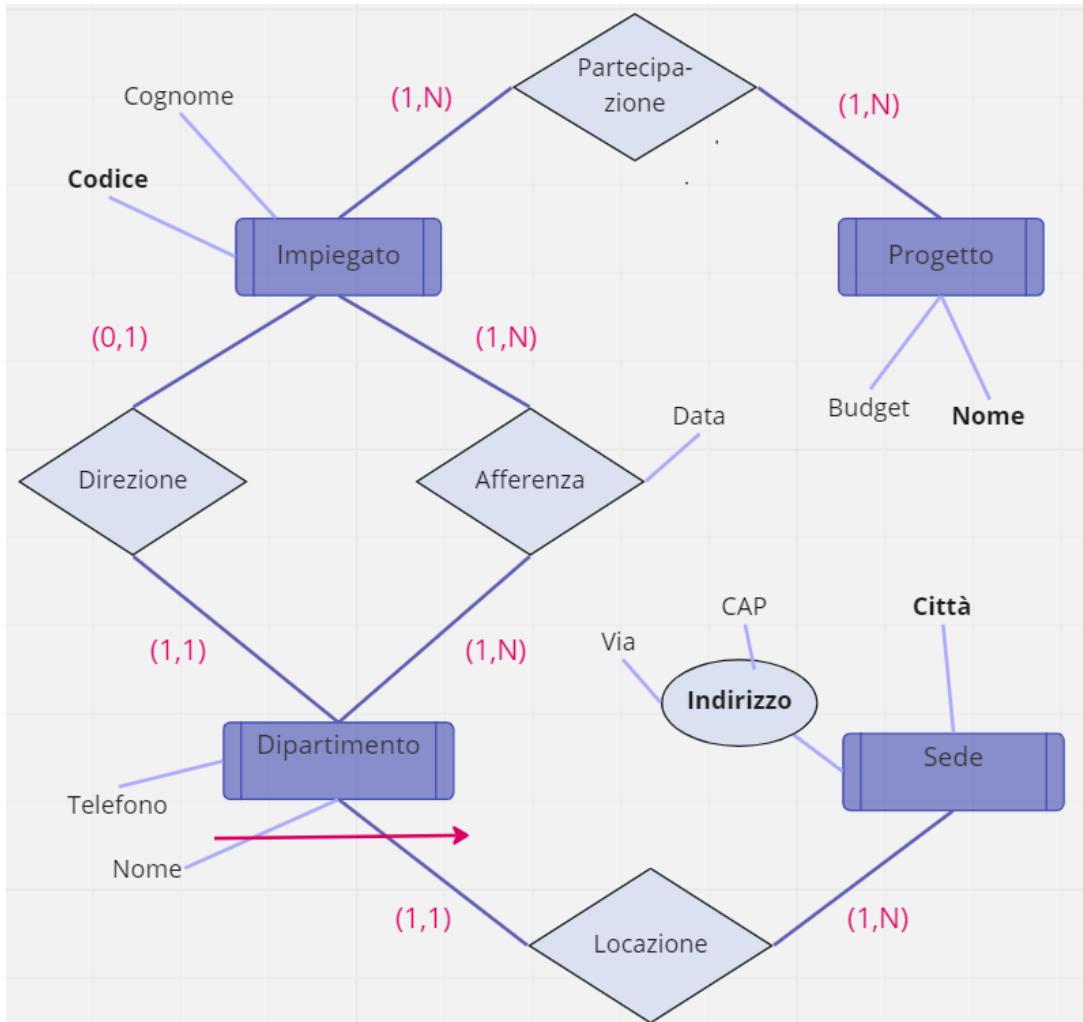
ESERCIZIO

Di ogni **impiegato** si conosce il **codice** e il **cognome**, a quali dipartimenti afferisce, ed eventualmente da quale **anno** ha cominciato ad afferirvi. Del **dipartimento** si conosce il **nome**, il **numero di telefono**, il **direttore**, e la **sede**. Possono esistere dipartimenti con lo stesso nome purché in sedi diverse. Di ogni **sede** si conosce l'**indirizzo** (**via**, **cap**) e la **città**. Gli impiegati partecipano a **progetti** di cui si conosce il **nome** e il **budget**.

Aggiungi i vincoli di cardinalità allo schema seguente:



- Portiamo dall'associazione **Direzione**, la cardinalità da **Impiegato** a **Direzione** sarà **(0,1)**, un impiegato non deve essere per forza un direttore, e in caso lo fosse può dirigere al massimo un dipartimento. Dall'altro lato invece dipartimento avrà un solo direttore, quindi **(1,1)**;
- Adesso consideriamo l'associazione **Afferenza**, un impiegato può appartenere a più dipartimenti, ma difficilmente non afferisce a nessun dipartimento, quindi mettiamo **(1,N)**, quando la specifica è ambigua bisogna specificare la nostra teoria. Dall'altro lato, in un dipartimento si avranno molti impiegati, quindi **(1,N)**;
- Possiamo all'associazione **Partecipazione**, un impiegato può partecipare a più progetti, e sembra che debbano parteciparci necessariamente, quindi **(1,N)**. Dall'altro lato ad un progetto partecipano più impiegati, quindi **(1,N)**;
- Infine, abbiamo **Locazione**, da Dipartimento avremo che un dipartimento si trova in un'unica sede quindi **(1,1)** e questo lo possiamo notare anche dal fatto che la chiave è costituita anche dalla chiave esterna. Dall'altro lato invece in una sede possiamo avere più dipartimenti (nel Palazzo c'è sia il dipartimento di Matematica che di Informatica), quindi **(1,N)**, difficilmente non esiste nessun dipartimento in una sede.



Bisogna stare attenti perché anche differenze apparentemente piccole in cardinalità e identificatori possono cambiare di molto il significato.

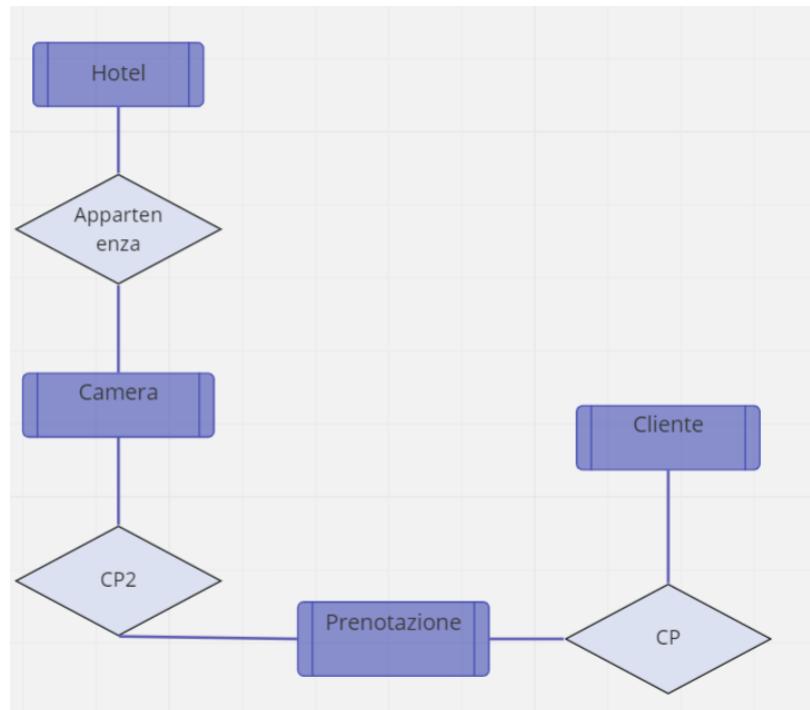
Quindi, ricapitolando, quando ci troviamo di fronte ad un esercizio in cui dobbiamo costruire lo schema ER, quali sono i **passi da seguire?**

1. **Individuare le entità e le associazioni;**
2. **Disegnare le entità (rettangoli), le associazioni (rombi), e collegarle;**
3. **Scrivere i vincoli di cardinalità delle associazioni (0,1), (0,N), (1,1), (1,N);**
4. Individuare gli **attributi** di entità e associazioni e aggiungerli allo schema (cerchietti bianchi);
5. Individuare le **chiavi** di ogni entità e segnarle nello schema (cerchietti neri);
6. Alla fine, ricontrillare il tutto. Lo schema deve riprodurre fedelmente quanto presente nella specifica. Non deve mancare nessuna informazione. Non ci devono essere informazioni non richieste!

ESERCIZIO

Si rappresenti lo schema ER di un archivio per la gestione delle prenotazioni delle camere di una catena di hotel. Di ogni **camera** si conosce il **numero**, il **piano** e l'**hotel** a cui appartiene. L'**hotel** ha un **nome**; se ne conosce l'**indirizzo** e il **telefono**. Le camere sono prenorate da **clienti**, di cui si conosce il **codice fiscale**, **nome**, **cognome**, e **telefono**. Lo stesso cliente può prenare più volte la stessa camera. Ogni **prenotazione** è identificata dalla **camera** e dalla **data di pernottamento**.

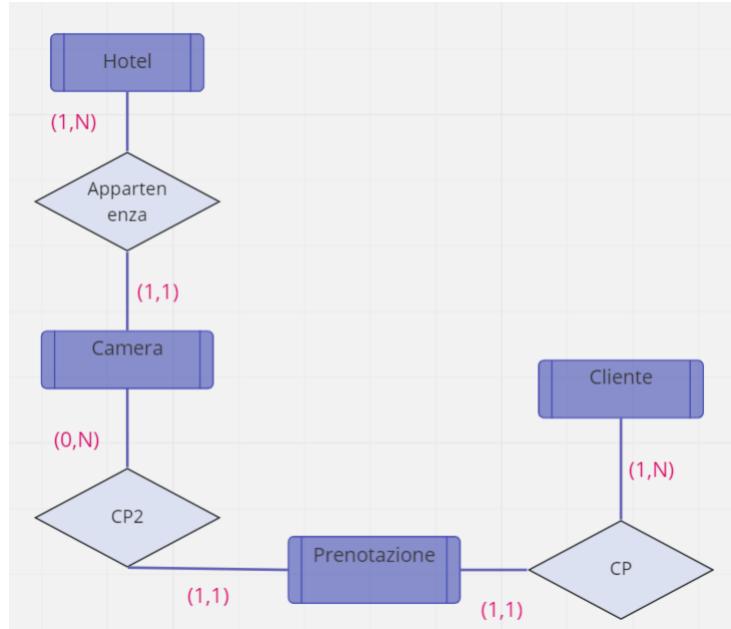
1 & 2:



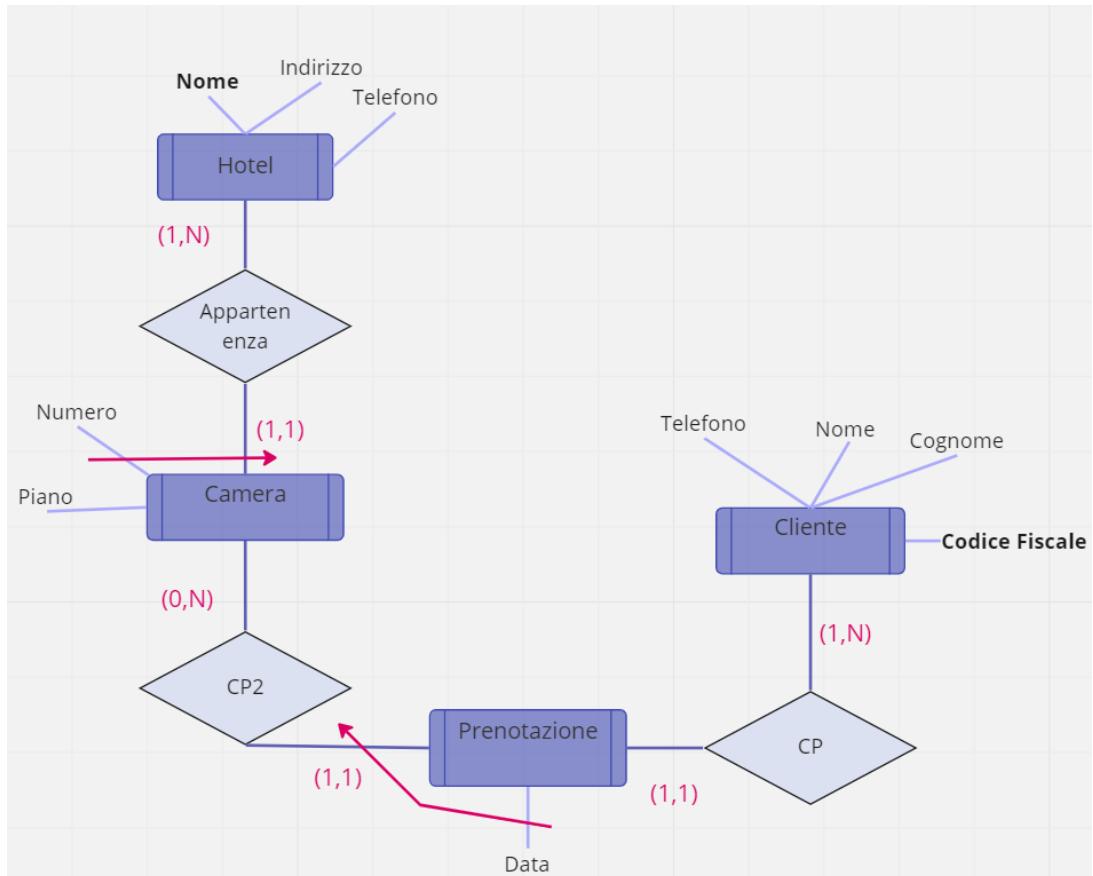
In questo caso **prenotazione** non va bene come associazione, perché ci possono essere delle coppie ripetute, in quanto un cliente può prenare più volte la stessa camera, per questo motivo l'associazione deve essere materializzata.

3:

- In un **Hotel** deve esserci almeno una **camera** fino ad N quindi **(1,N)**, una **camera** è appartenuta ad un solo **hotel** quindi **(1,1)**;
- Una **Camera** può essere prenata più volte (in quanto vogliamo tenere un archivio, quindi nel tempo può essere prenata più volte) ma anche 0 volte quindi **(0,N)**, una **prenotazione** può essere fatta su una sola **camera**, in quanto è proprio identificata dalla **camera** quindi **(1,1)**;
- Un **Cliente** può prenare più camere, ma un minimo di 1 perché se no il cliente non sarebbe tale quindi **(1,N)**, la **prenotazione** deve riferirsi ad uno e un solo cliente (colui che ha prenato) quindi **(1,1)**.

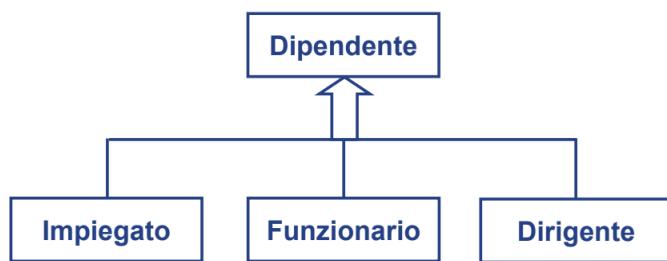


4 & 5:



- Gli attributi di **Hotel** sono: **nome** (chiave primaria), **indirizzo** e **telefono**;
- Gli attributi di **Camera** sono: **piano** e **numero**, la chiave primaria è composta dal **numero** e **dall'hotel a cui appartiene**;
- Gli attributi di **Prenotazione** sono la **data**, e la chiave primaria è composta dalla **data** e dalla **camera su cui è stata effettuata**;
- Gli attributi di **Cliente** sono il **codice fiscale** (chiave primaria), il **nome**, il **cognome** e il **telefono**.

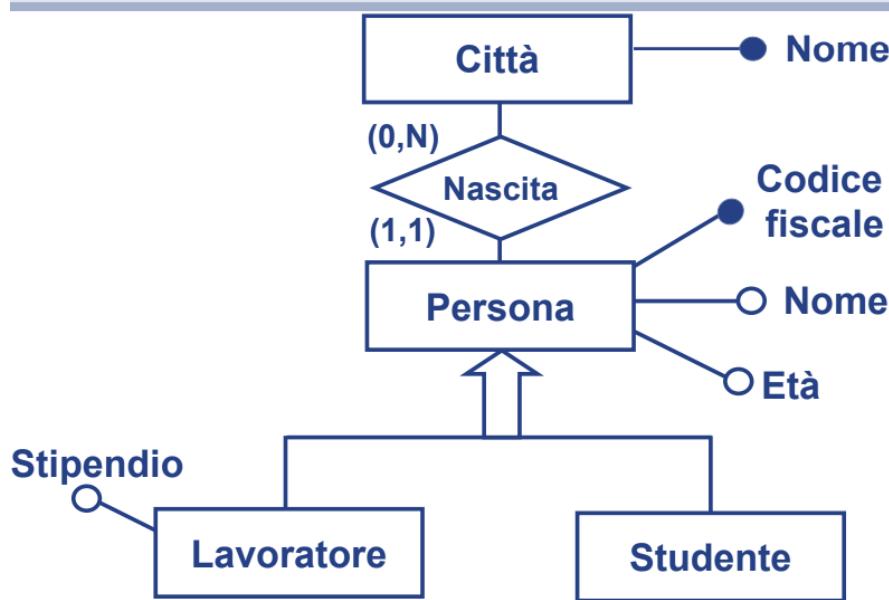
La **generalizzazione** mette in relazione una o più entità E1, E2, ..., En con una entità E, che le comprende come casi particolari, E è **generalizzazione** di E1, E2, ..., En dove E1, E2, ..., En sono **specializzazioni** (o sottotipi) di E.



In questo esempio dipendente è una generalizzazione di altre tre entità più specifiche.

Se E (genitore) è generalizzazione di E1, E2, ..., En (figlie):

- ogni proprietà di E è **significativa** per E1, E2, ..., En, se ogni dipendente ha un nome, allora anche impiegato, funzionario hanno un nome, ma non è vero il contrario;
- ogni occorrenza di E1, E2, ..., En è **occorrenza anche di E**, ogni istanza di **Impiegato** è istanza di **Dipendente**, ma non è vero il contrario;

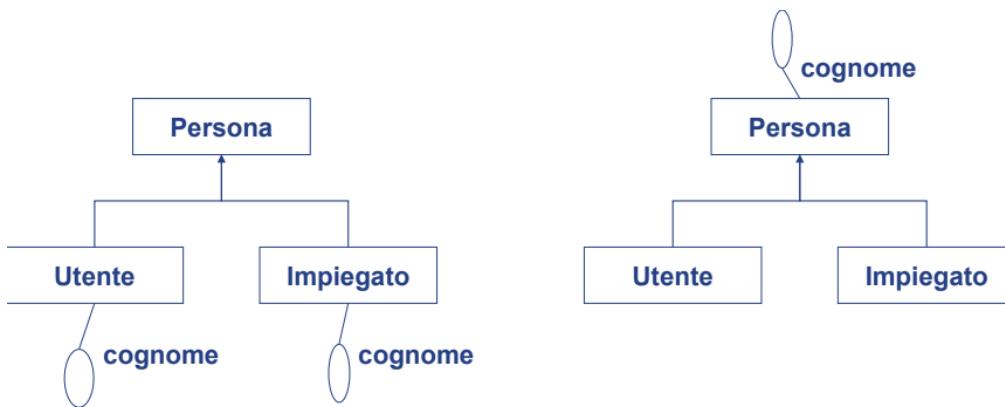


In questo caso **Persona** è una generalizzazione, e le figlie sono il **Lavoratore** che eredita gli attributi di Persona più un suo attributo stipendio; invece, **Studente** eredita semplicemente gli attributi del genitore. Se studente avesse un altro attributo che fosse chiave, le sue chiavi sarebbero l'unione della sua + quella del genitore.

Quindi tutte le proprietà (attributi, associazioni, altre generalizzazioni) dell'entità genitore vengono ereditate dalle entità figlie e non rappresentate esplicitamente, e l'entità genitore non eredita dalle figlie!

ESERCIZIO

Quale dei due è corretto?

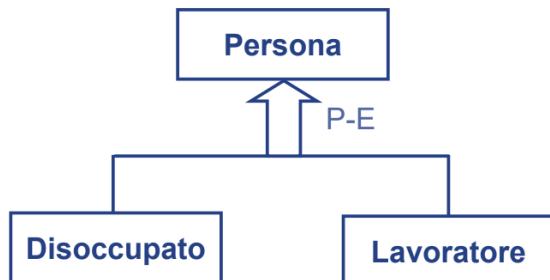


In questo caso è **corretto il secondo** perché ogni persona ha un cognome, e quest'ultimo viene ereditato da Utente e Impiegato. Se ci sono degli attributi comuni fra i figli, è sempre meglio metterli come attributi del genitore in modo tale che non vengano ripetuti.

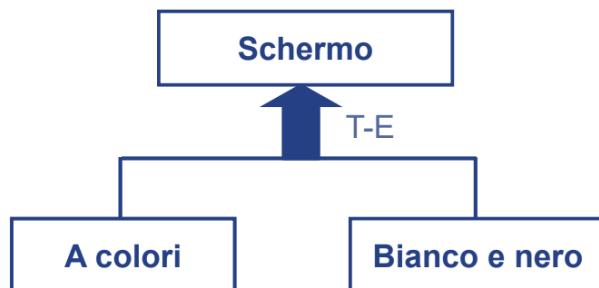
Esistono **diversi tipi** di generalizzazione:

- **Totale**: se ogni occorrenza dell'entità genitore è occorrenza di almeno una delle entità figlie (se tutte le persone sono utenti, impiegati, o entrambi), altrimenti è **Parziale** (esistenza di persone generiche);
- **Esclusiva**: se ogni occorrenza dell'entità genitore è occorrenza di al più una delle entità figlie (se una persona è utente non può anche essere impiegato), altrimenti è **Sovrapposta**.

Che tipo di generalizzazione?

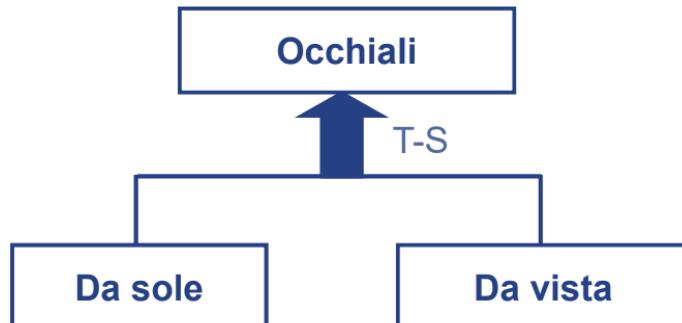


È **parziale** perché nelle persone vi possono ad esempio essere dei bambini che non possono né essere disoccupati né lavoratori, ed è **esclusiva** perché un disoccupato non può essere un lavoratore, e viceversa.



È **totale** perché uno schermo può essere solo a colori o bianco e nero, non ci possono essere schermi di altro tipo, ed è **esclusiva** perché lo schermo può essere solo in una determinata maniera contemporaneamente.

La **freccia bianca** in generale indica una generalizzazione **parziale**, e quella **nera** invece una **totale**, ma nell'esame basta inserire l'acronimo T-E.

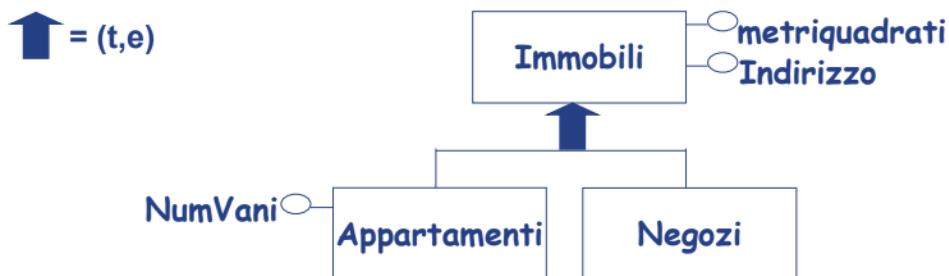


È **totale** perché se pensiamo all'ottico possiamo acquistare solo questi due tipi di occhiali, ed è **sovrapposta** perché possono esistere occhiali da vista che sono anche da sole.

Possono esistere **gerarchie a più livelli** e multiple generalizzazioni allo stesso livello; quindi, un'entità può essere inclusa in più gerarchie, come genitore e/o come figlia.

Se una generalizzazione ha solo un'entità figlia si parla di **sottoinsieme**. Il genitore di una generalizzazione totale può non avere identificatore, purché tutte le figlie abbiano una chiave.

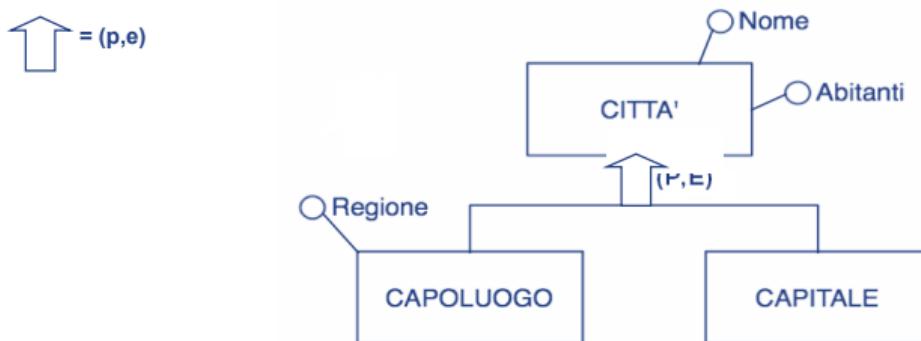
ESERCIZI V/F:



Considerato lo schema ER in figura, rispondere alle seguenti domande:

1. Un appartamento può essere usato come negozio;
 2. Un negozio non è necessariamente un immobile;
 3. Un immobile è necessariamente un appartamento o un negozio;
 4. Le istanze della entità negozio non hanno attributi;
 5. Tra le proprietà di Negozio c'è il numero di vani.
-

1. Falso, perché è esclusiva, quindi non possono essere entrambi;
2. Falso, perché ogni istanza delle figlie è anche istanza del genitore, quindi il negozio è un immobile;
3. Vera, perché è totale, non posso avere immobili che non sono né appartamenti ne negozi;
4. Falso, perché eredita gli attributi del genitore;
5. Falso, perché le figlie non ereditano gli attributi fra di loro.



Si consideri la seguente gerarchia di generalizzazione: quale delle seguenti affermazioni è corretta?

1. Ogni città è capoluogo o capitale;
 2. Una capitale è anche capoluogo;
 3. Un capoluogo può essere anche capitale;
 4. Ogni capitale è caratterizzata dalla regione di appartenenza;
 5. Ogni capitale è caratterizzata dal numero di abitanti;
 6. Se una città è capoluogo non può essere capitale.
-

1. Falso, perché è parziale;
2. Falso, perché è esclusiva e non sovrapposta;
3. Falso;
4. Falso, lo è il capoluogo che ha l'attributo regione;
5. Vero, perché lo eredita dal genitore;
6. Vera.

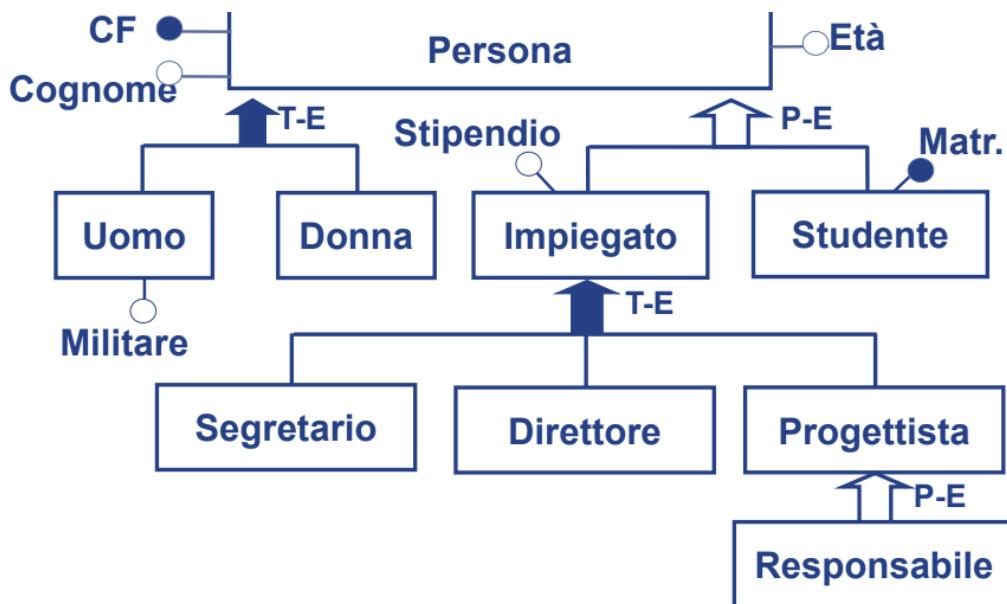
ESERCIZIO:

Le **persone** hanno CF, cognome ed età; gli **uomini** anche la **posizione militare**; gli **impiegati** hanno lo stipendio e possono essere **segretari, direttori o progettisti** (un progettista può essere anche responsabile di progetto); gli **studenti** (che non possono essere impiegati) hanno un numero di matricola; esistono persone che non sono né impiegati né studenti (ma i dettagli non ci interessano).

Allora partiamo dalla creazione di **Persona**, quest'ultima ha come attributi il codice fiscale, il cognome e l'età. dopodiché dobbiamo inserire gli Uomini che sono persone, quindi avremo come figlie di Persona **Uomo** e **Donna**, questo tipo di generalizzazione deve essere totale in quanto una persona deve essere necessariamente o uomo o donna (Riboni ha paura dei non-binary), e deve essere esclusiva perché una persona non può essere sia uomo che donna, inoltre l'entità uomo avrà come attributo aggiuntivo la **posizione militare**.

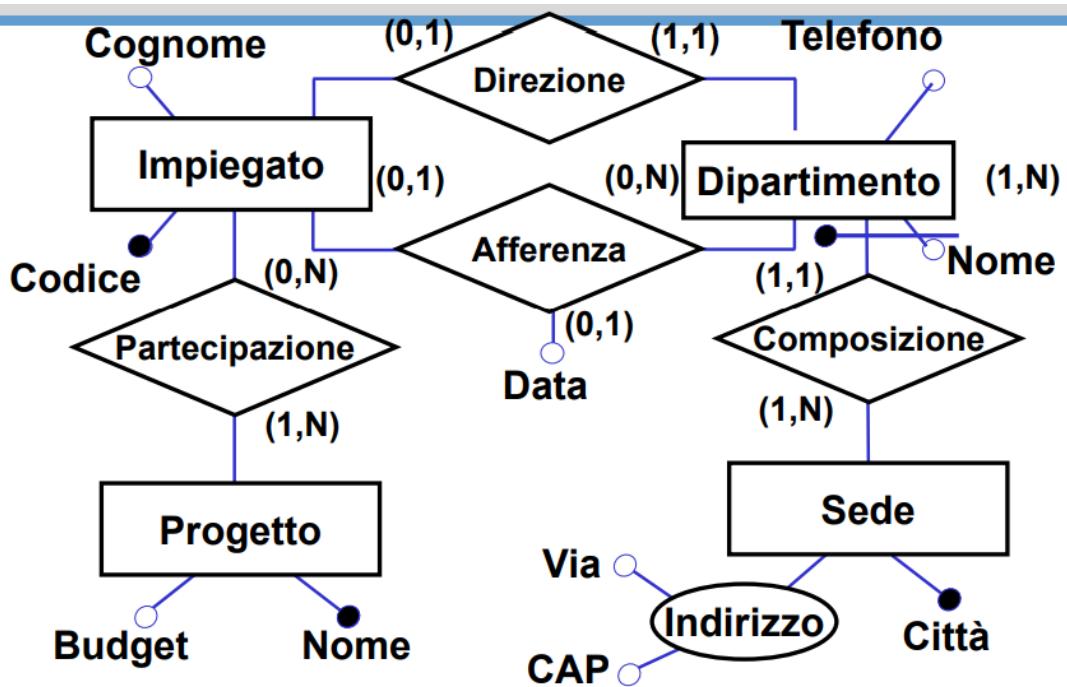
Un'altra distinzione che possiamo fare è **Impiegato** e **Studente**, che sono sicuramente delle persone, questa generalizzazione è parziale perché una persona non è detto che sia per forza o un impiegato o uno studente ed è esclusiva perché un impiegato non può essere uno studente e viceversa, inoltre aggiungiamo l'attributo **stipendio** in Impiegato, e la **matricola** in Studente.

Inoltre, un impiegato può essere un **Segretario**, un **Direttore** o un **Progettista**; quindi, anche questa sarà una generalizzazione totale ed esclusiva. Infine, il progettista può essere anche un responsabile e il loro collegamento è parziale ed esclusivo, **che è l'unica possibilità quando si ha un unico figlio**.



La **documentazione** associata agli schemi concettuali è composta da:

- Dizionario dei dati:
 - Entità;
 - Associazione;
- Vincoli non esprimibili in ER (potrebbe chiedere di farlo all'esame).



Consideriamo questo esempio, il dizionario dei dati di questo schema è:

Entità	Descrizione	Attributi	Identificatore
Impiegato	Dipendente dell'azienda	Codice, Cognome, Stipendio	Codice
Progetto	Progetti aziendali	Nome, Budget	Nome
Dipartimento	Struttura aziendale	Nome, Telefono	Nome, Sede
Sede	Sede dell'azienda	Città, Indirizzo	Città

Un esempio di possibili vincoli non esprimibili in ER può essere:

Vincoli di integrità sui dati
(1) Il direttore di un dipartimento deve afferire a tale dipartimento
(2) Un impiegato non deve avere uno stipendio maggiore del direttore del dipartimento al quale afferisce
(3) Un dipartimento con sede a Roma deve essere diretto da un impiegato con più di dieci anni di anzianità
(4) Un impiegato che non afferisce a nessun dipartimento non deve partecipare a nessun progetto

PROGETTAZIONE LOGICA

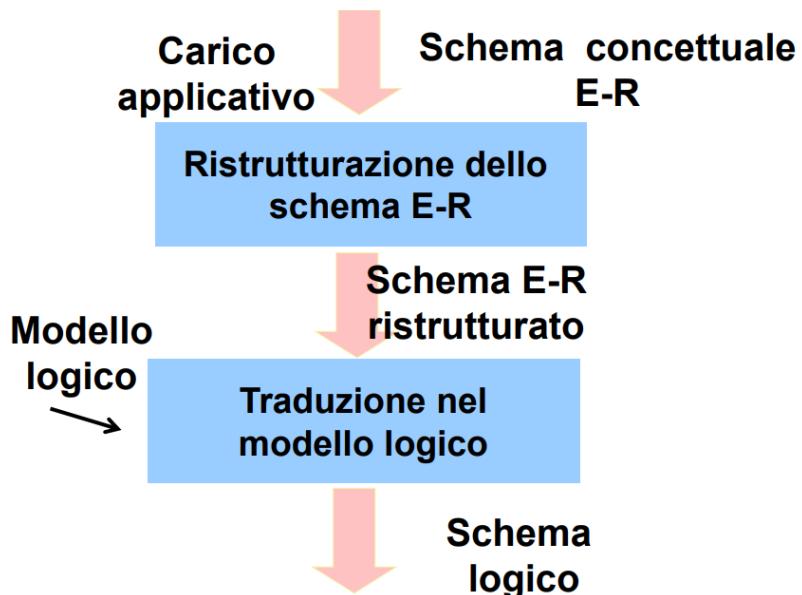
L'obiettivo della **progettazione logica** è quello di tradurre lo schema concettuale in uno schema logico che rappresenti gli stessi dati in maniera **corretta ed efficiente**, non tutti i costrutti di ER possono essere tradotti direttamente in uno schema relazionale (ad esempio non è possibile tradurre le gerarchie).

I dati che riceve in ingresso sono:

- Lo schema concettuale;
- Quale modello logico utilizzare;
- Le informazioni sul carico applicativo, per ottimizzare in base l'accesso ai dati (non lo vedremo nel corso).

In output avremo:

- Lo schema logico;
- La documentazione associata.



Prima di definire effettivamente le tabelle dobbiamo fare **un'attività di ristrutturazione** dello schema, questa attività si compone di 4 fasi:

1. Analisi delle ridondanze;
2. Eliminazione delle generalizzazioni;
3. Eliminazione di attributi multivaleure;
4. Scelta degli identificatori primari.

Una **ridondanza** in uno schema E-R è una **informazione significativa ma derivabile da altre**, in questa fase si decide se eliminare le ridondanze eventualmente presenti o mantenerle (o anche di introdurne di nuove).

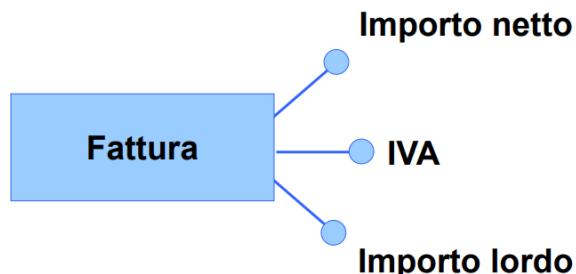
I vantaggi sono la **semplificazione delle interrogazioni**, perché se ho un'informazione ripetuta più volte è possibile che questo mi renda più facile estrarre la informazione. Gli svantaggi sono:

- **appesantimento degli aggiornamenti**: bisogna aggiornare quella informazione ovunque si trovi;
- **maggior occupazione di spazio**: più informazioni da memorizzare;
- **possibili inconsistenze**: bisogna essere sicuri che quel valore sia effettivamente giusto rispetto alle informazioni da cui si potrebbe derivare;

Le **forme di ridondanza** in uno schema ER possono derivare da:

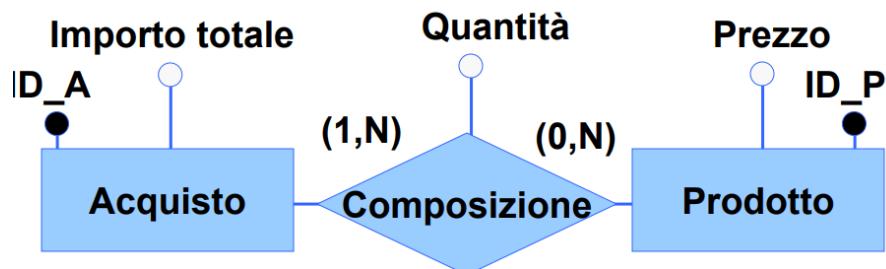
- Attributi derivabili:
 - da altri attributi della stessa entità (o associazione);
 - da attributi di altre entità (o associazioni).
- Associazioni derivabili dalla composizione di altre (più in generale: cicli di associazioni);

Attributo derivabile

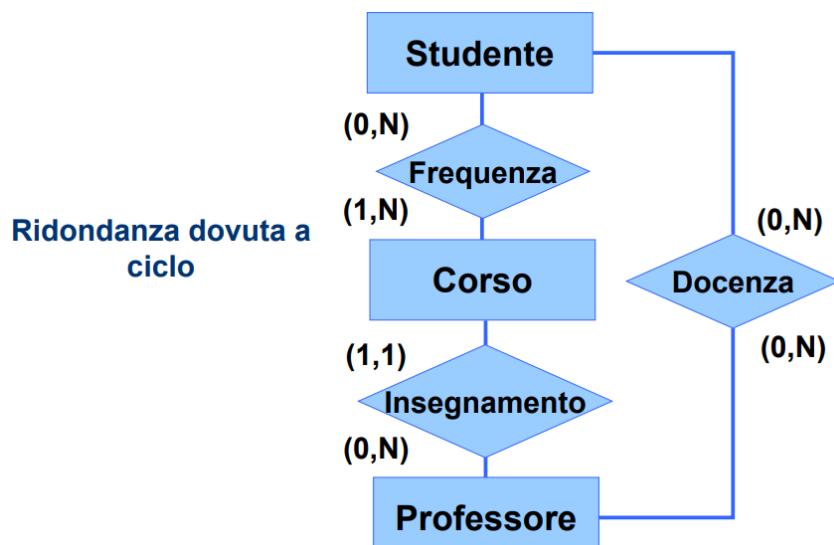


In questo esempio l'importo lordo è dato dall'importo netto + iva; quindi, si potrebbe decidere di rimuoverlo perché è facilmente derivabile. Ma nel caso mi servisse immediatamente quel valore è meglio lasciarlo così.

Attributo derivabile da altra entità



In questo esempio ho degli acquisti di prodotti, ogni prodotto lo posso acquistare in diverse quantità. L'attributo derivabile è l'**importo totale** che è dato dalla sommatoria di quantità * prezzo, anche in questo caso scegliere se lasciare l'attributo importo totale o derivarlo successivamente dipende da quali tipo di query vengono richieste più spesso.



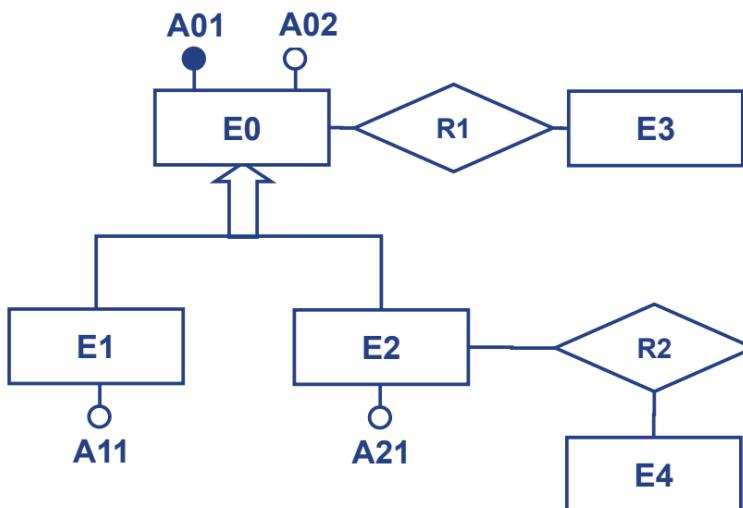
In questo esempio lo studente frequenta un corso che è insegnato da un professore, e poi abbiamo un'associazione aggiuntiva fra studente e professore, che è comunque ottenibile quindi è un'informazione ridondante.

In generale **non tutte le ridondanze vanno eliminate**, ma se si mantengono delle ridondanze occorre motivarne l'utilità. Ad esempio: "nello schema precedente mantengo Docenza perché molte query riguardano direttamente quella informazione.

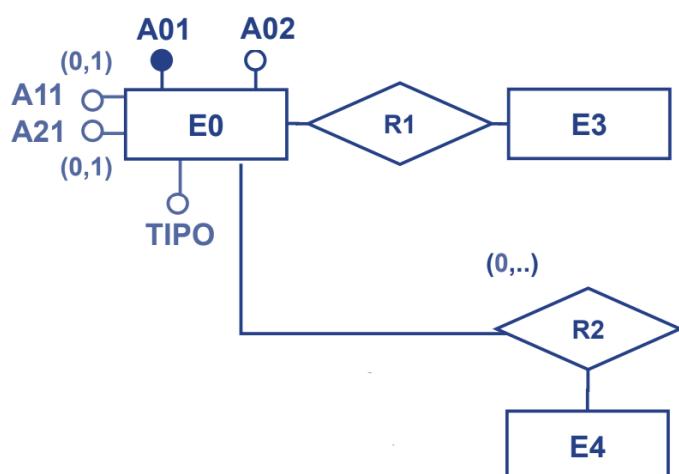
Il **modello relazionale** (ricordiamo che è il modello logico) **non può rappresentare direttamente le generalizzazioni**, invece, entità e associazioni sono direttamente rappresentabili. **Si eliminano perciò le gerarchie**, sostituendole con entità e associazioni. Ci possono essere 3 possibilità:

1. accorpamento delle figlie della generalizzazione nel genitore;
2. accorpamento del genitore della generalizzazione nelle figlie;
3. sostituzione della generalizzazione con associazioni.

Consideriamo questo schema ER:



La **prima tecnica** consiste nell'accorpamento delle figlie della generalizzazione nel genitore, conviene quando c'è poca distinzione fra figlie e padre (molti attributi in comune, pochi delle singole figlie):

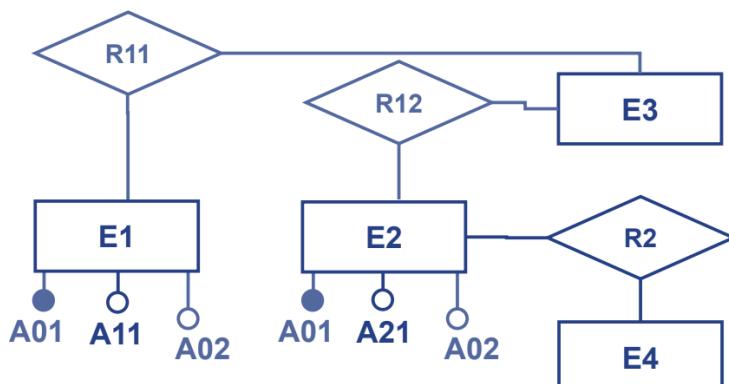


Rimuoviamo quindi le entità E1 e E2, e le accorpiamo a E0 (genitore), unendo a quest'ultima gli attributi A11 e A21 delle due entità. Infine, si aggiunge un attributo **Tipo** per specificare quale sottoclasse appartenesse a quella istanza.

Per specificare che sono opzionali utilizziamo **(0,1)**.

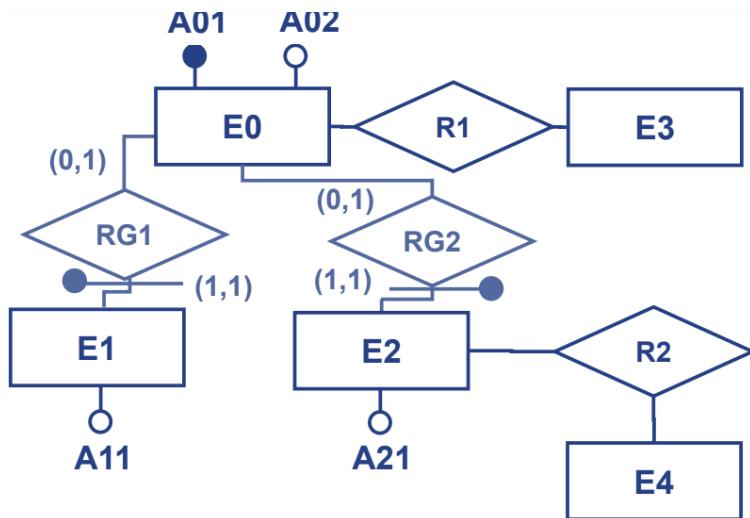
La **seconda tecnica** consiste nell'acorpamento del genitore della general. nelle figlie:

- conviene se c'è molta distinzione tra figlie/padre;
- è possibile solo con generalizzazione totale.



Inseriamo gli attributi del genitore nelle figlie, e creiamo le associazioni fra le entità figlie e le entità con cui precedentemente era collegato il genitore.

La **terza tecnica** è quella che effettua una sostituzione della generalizzazione con delle associazioni, conviene se c'è molta distinzione tra le figlie.

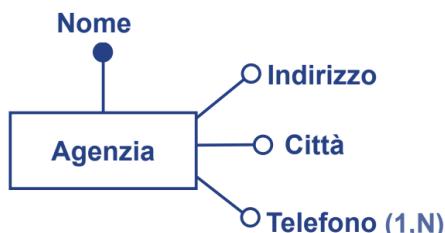


In questo caso lasciamo tutte le entità e al posto della generalizzazione creiamo delle associazioni con ogni entità figlia.

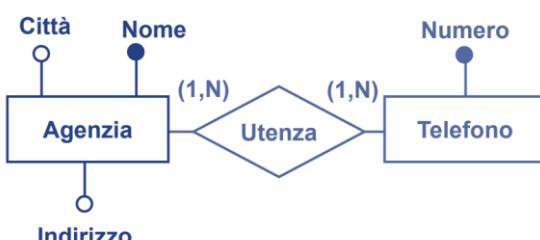
Per far questo le figlie avranno come chiave primaria la chiave esterna del genitore, e la cardinalità delle associazioni sarà **(1,1)** dalla figlia verso l'associazione perché necessariamente una figlia deve essere anche un genitore (e poi è anche la regola per poter utilizzare la chiave esterna).

Invece dal genitore all'associazione la cardinalità sarà **(0,1)** perché ci possono essere istanze di E0 che non sono istanze né di E1 né di E2.

La terza fase della ristrutturazione consiste **nell'eliminazione di attributi multivalue**:



In questo caso l'agenzia può avere più attributi telefono, nel modello relazionale **non è possibile tradurre questa informazione**, in quanto in una tabella nelle colonne può stare un unico valore. Per poter tradurre questo si può trasformare l'attributo telefono in un'entità:



L'ultimo passo è la **scelta degli identificatori primari**, è un'operazione indispensabile per la traduzione nel modello relazionale. I criteri sono:

- assenza di opzionalità;
- semplicità;
- utilizzo nelle operazioni più frequenti o importanti.

Se nessuno degli identificatori soddisfa i requisiti visti? Si introducono nuovi attributi (**codici**) contenenti valori speciali generati appositamente per questo scopo.

ESERCIZIO

Si rappresenti la base di dati di un archivio per la gestione di vendite all'asta di oggetti. Gli **oggetti** hanno un **nome**, e si conosce il **venditore** a cui appartengono, il **prezzo di partenza**, e le date di inizio e fine asta.

Oggetti diversi possono avere nomi uguali, ma non se sono dello stesso venditore. Di ogni **venditore** si conosce CF, nome, cognome e telefono. Di ogni **offerente** si conosce CF, nome, cognome e indirizzo di spedizione. Di ogni **offerta** si conosce l'offerente, l'oggetto per cui è stata presentata l'offerta, la data, e la cifra. Lo stesso offerente può fare più offerte per lo stesso oggetto, purché la cifra sia diversa.

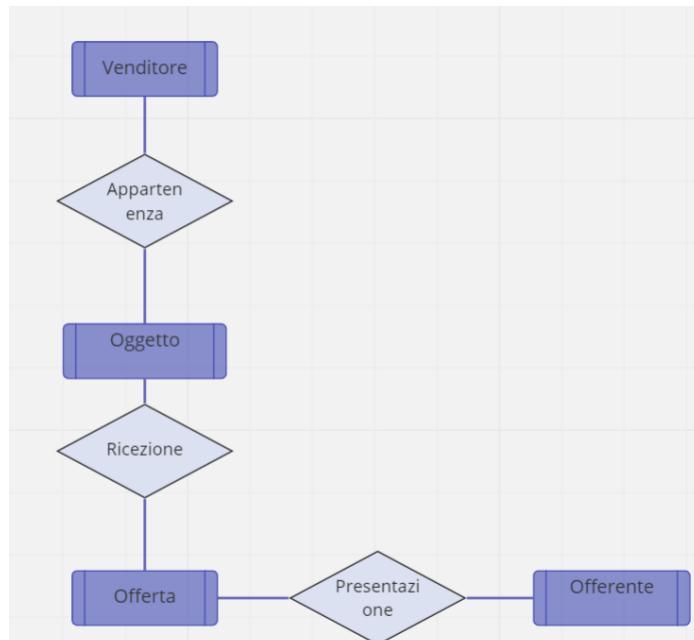
Iniziamo a trovare le **entità** e le **associazioni**:

- Venditore;
- Offerente;
- Oggetto;

-
- Il **venditore** possiede degli oggetti, quindi vi è un'associazione fra venditore e oggetto;
 - L'**offerta** viene effettuata da un offerente verso un determinato oggetto; quindi, vi è un'associazione fra queste entità;
 - L'elemento che viene messo in offerta è un **oggetto**, quindi vi è un'associazione fra oggetto e offerta.

Non è necessario **materializzare** l'associazione fra venditore e oggetto perché non vogliamo avere uno storico (quindi nessuna coppia ripetuta), neanche per quella fra oggetto e offerta.

Però quella fra **offerente e oggetto** sì perché lo stesso offerente può fare più offerte per lo stesso oggetto; quindi, ci potrebbero essere delle coppie ripetute.

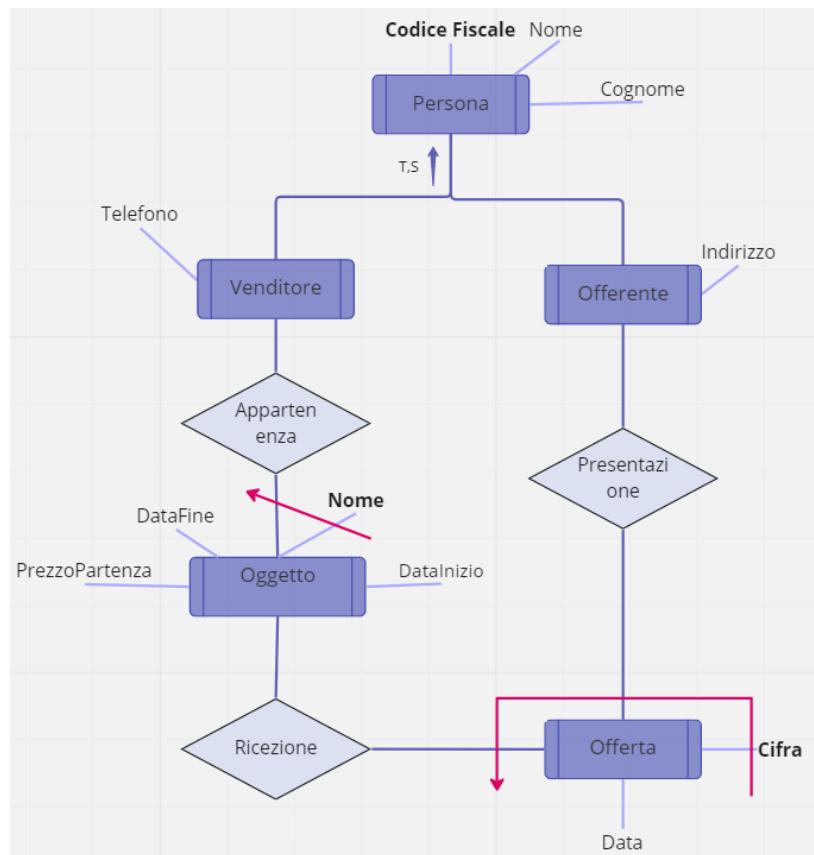


Il passo successivo è quello di [introdurre gli attributi](#), possiamo però notare che gli attributi di venditore e offerente sono molto simili; quindi, potremmo aggiungere una [generalizzazione Persona](#) con figlie Venditore e Offerente.

La generalizzazione sarà **totale** perché nella nostra base di dati una persona può essere un venditore o un offerente e nient'altro, e può essere [sovraposta](#) perché un venditore può anche essere un offerente (ad es. Ebay.)

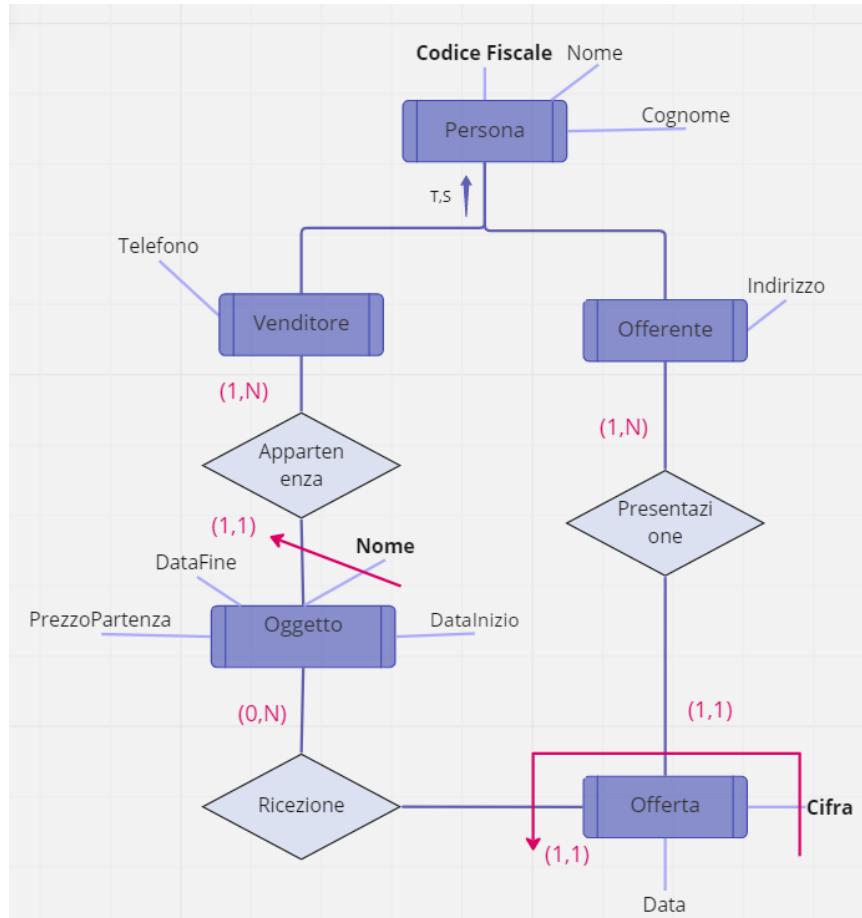
Dobbiamo trovare anche le chiavi:

- Per Oggetto scegliamo la chiave **<Nome,Oggetto>**, in quanto si conosce il nome del venditore a cui appartengono;
- Per Persona scegliamo il **codice fiscale**, per Venditore e Offerente non è necessaria perché la ereditano da Persona;
- Per Offerta scegliamo la chiave **<Cifra,Oggetto,Offerente>**, in quanto si conosce l'offerente, e lo stesso offerente può fare più offerte per lo stesso oggetto, purché la cifra sia diversa.



Infine, troviamo i [vincoli di cardinalità](#):

- Da Venditore verso appartenenza, è **(1,N)** perché per essere tale deve avere almeno un oggetto in vendita, e da Oggetto invece è **(1,1)** basta pensare che vi è la chiave esterna;
- Da Offerente a Offerta è **(1,N)** per lo stesso motivo di venditore; invece, da offerta a offerente è **(1,1)** sempre per il vincolo di chiave esterna e anche perché un offerta è fatta esattamente da un offerente;
- Da Oggetto a Offerta è **(0,N)** perché ci possono essere oggetti in vendita senza offerta, o più offerte per lo stesso oggetto, da Offerta è **(1,1)** sempre per il vincolo di chiave esterna.



Esistono dei vincoli che non sono esprimibili:

- La data inizio deve essere sicuramente prima di data fine;
- Il prezzo offerto per uno stesso oggetto deve essere sempre maggiore.

Ristrutturare lo schema ER precedente, passi:

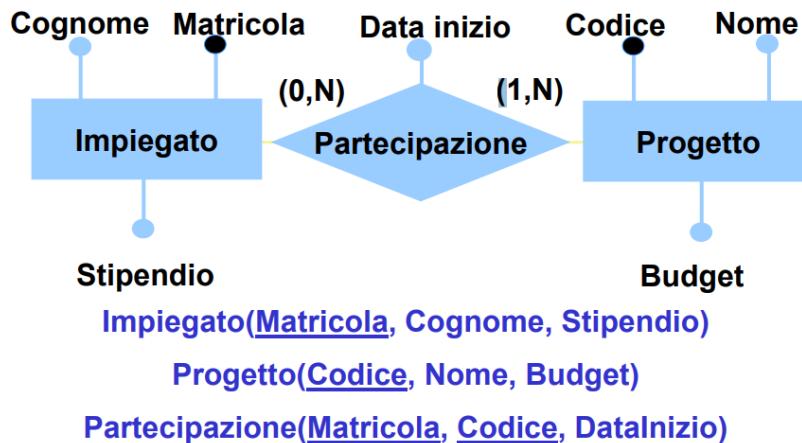
1. Analisi delle ridondanze;
2. Eliminazione delle generalizzazioni;
3. Eliminazione di attributi multivalore;
4. Scelta degli identificatori primari.

1. Non vi sono ridondanze;
2. Dobbiamo trasformare la generalizzazione **Persona**, in questo caso è meglio accoppare le due figlie nel genitore perché hanno pochi attributi caratteristici. Quindi aggiungiamo l'attributo **Tipo** che specifica se è un venditore o un offerente o entrambi, e aggiungiamo gli attributi delle due figli come optionali;
3. Non vi sono attributi multivalore;
4. Ogni entità ha una sua chiave, quindi, non c'è bisogno di scegliere.

L'idea di base per la [traduzione verso il modello relazionale](#) è:

- Le entità diventano relazioni (tabelle) sugli stessi attributi;
- Le associazioni possono diventare o meno relazioni, in generale lo diventano quando abbiamo un'associazione N a N:
 - se diventano relazioni hanno come chiave gli identificatori delle entità coinvolte.

Entità e associazioni molti a molti:



In questo esempio partecipazione è un'associazione molti a molti (cardinalità massima N in entrambe le parti), quindi dovrà diventare una tabella.

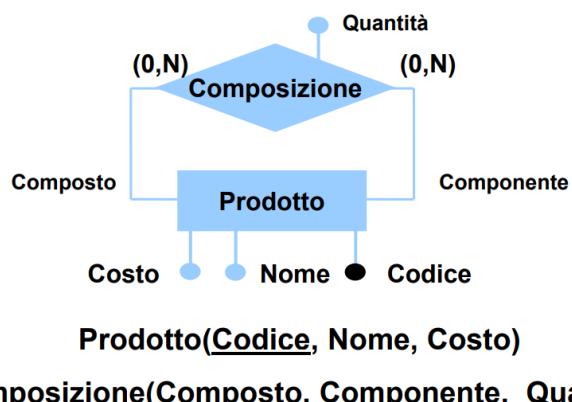
Con vincoli di integrità referenziale fra:

- **Matricola** in **Partecipazione** e (la chiave di) **Impiegato**;
- **Codice** in **Partecipazione** e (la chiave di) **Progetto**.

In questo caso un impiegato può partecipare solo una volta allo stesso progetto (non ci possono essere coppie ripetute), perché la partecipazione è identificata dalla matricola dell'impiegato e dal codice del progetto.

Per rendere la tabella più comprensibile solitamente si utilizzano **nomi più espressivi** per gli attributi della chiave della relazione che rappresenta l'associazione. In questo esempio le chiavi di **Partecipazione** potrebbero essere **Impiegato** e **Progetto**, al posto di matricola e codice.

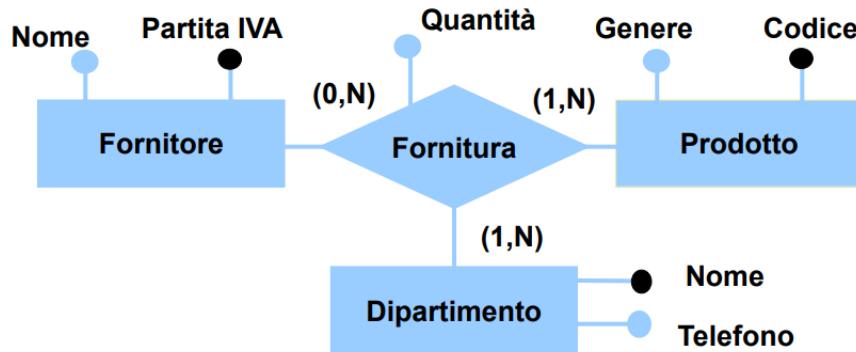
Associazioni ricorsive molti a molti:



Per differenziare i ruoli in una associazione ricorsiva asimmetrica molti a molti, si inseriscono i campi all'interno dell'associazione, come chiave primaria.

In questo esempio **Composto** e **Componente** sono codici del **Prodotto**.

➤ Associazioni n-arie molti a molti:



In questo esempio l'associazione fornitura conterrà al suo interno la chiave di Fornitore, di Prodotto e di Dipartimento. Quindi in generale all'interno di un associazione vi sono tutte le chiavi delle relazioni (entità) a cui è collegata.

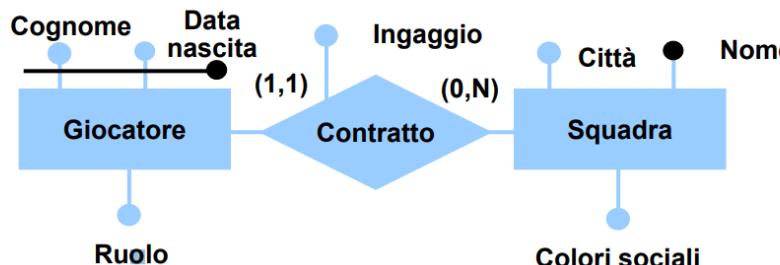
Se pensassimo allo stesso esempio ma con il collegamento fra fornitore e fornitura con cardinalità (0,1), cioè che lo stesso fornitore può fare al massimo una fornitura:

- Avremo la tabella Fornitura(P_IVA, Codice, Nome Dip, Quantità), e P_IVA deve essere UNIQUE per rispettare la condizione sopracitata;

In generale per comporre un'associazione n-aria molti a molti basta una coppia che lo sia.

➤ Associazioni uno a molti:

Associazioni uno a molti



Giocatore(Cognome, DataNasc, Ruolo, Squadra, Ingaggio)
Squadra(Nome, Città, ColoriSociali)

In questo caso l'associazione Contratto viene inglobata dall'entità Giocatore, cioè **dall'entità che ha come cardinalità 1**. Con vincolo di integrità referenziale fra **Squadra** in Giocatore e la chiave di Squadra.

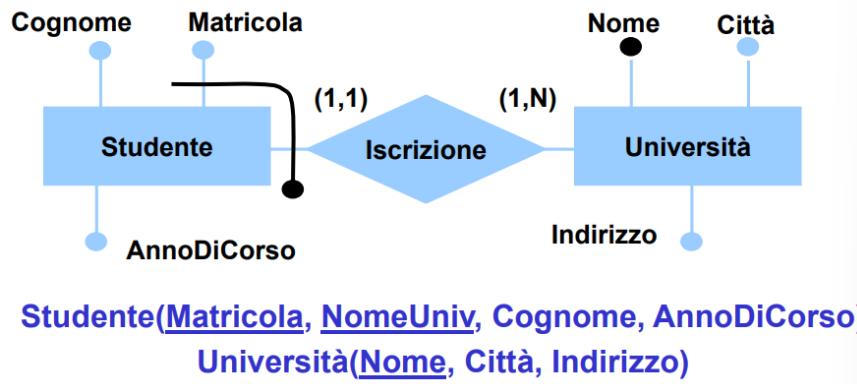
Se la cardinalità minima della associazione è **0**, allora Squadra e Ingaggio in Giocatore deve ammettere valore nullo. Questo si può fare anche se Squadra, ad esempio, ha un vincolo di integrità referenziale con Nome in Squadra (non chiave), l'importante è che Nome non sia NULL.

In generale la traduzione riesce a rappresentare efficacemente la cardinalità minima della partecipazione che ha **1 come cardinalità massima**:

- 0 : valore nullo ammesso;
- 1: valore nullo non ammesso.

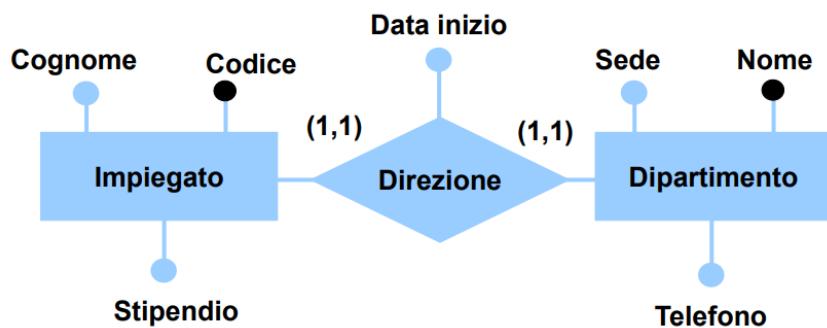
Se voglio indicare attributo opzionale uso il simbolo *****, come ad es: Giocatore(Cognome, DataNasc, Ruolo, Squadra*, Ingaggio*). In questo caso il vincolo tra Gioc. e Contratto sarebbe (0,1).

☞ Entità con identificazione esterna:



Con vincolo di integrità referenziale fra NomeUniv in Studente e Nome in Università.

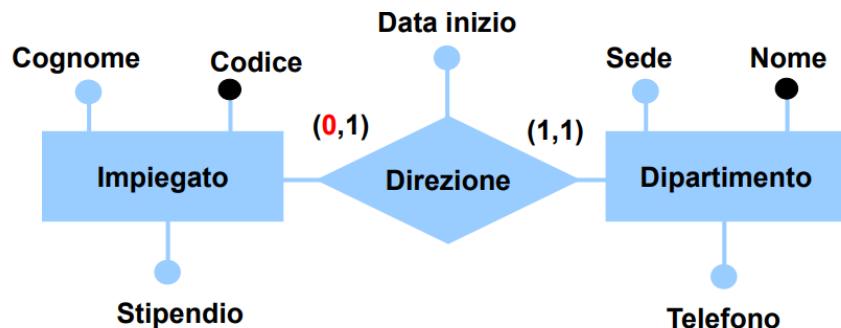
☞ Associazioni uno a uno:



Ci possono essere diverse possibilità in questo caso:

- fondere Direzione da una parte o dall'altra;
- fondere tutto? Cioè mettere tutto in una tabella Direzione.

Solitamente la possibilità privilegiata è la 1, quindi si sceglie una delle due relazioni in cui inserire i valori dell'associazione:

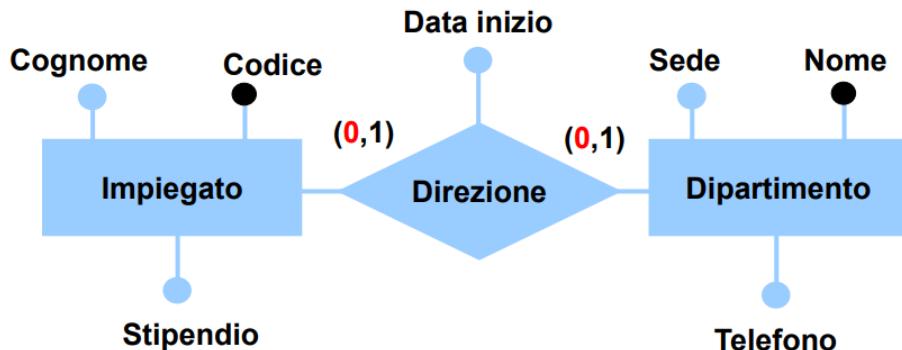


Impiegato (Cognome, Codice, Stipendio)

Dipartimento (Nome, Sede, Telefono, Direttore, InizioD)

In questo esempio si ha come cardinalità da Impiegato (0,1), per questo motivo si inserisce direzione in Dipartimento, questo permette di non avere alcun valore nullo. Non sarebbe scorretto inserirli in Impiegato, ma se si può è meglio evitare l'inserimento di valori NULL.

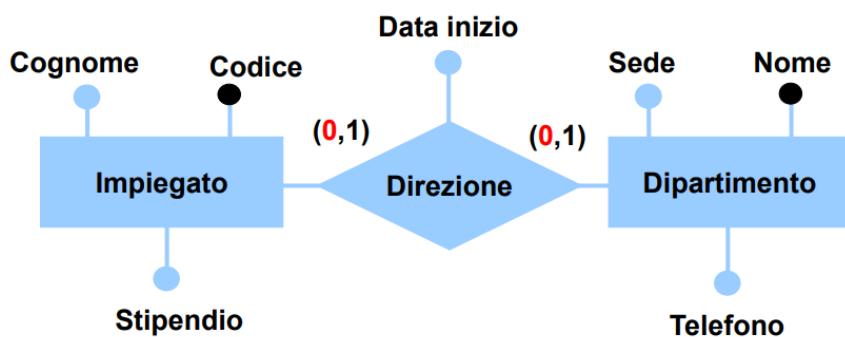
Bisogna avere anche come vincolo che Direttore sia **UNIQUE**, perché altrimenti lo stesso Impiegato potrebbe essere direttore di più Dipartimenti.



Impiegato (Codice, Cognome, Stipendio)

Dipartimento (Nome, Sede, Telefono, **Direttore*, **InizioD***)**

In questo caso abbiamo entrambe le cardinalità minime 0, per questo motivo gli attributi **Direttore** e **InizioD** devono essere opzionali, quindi possono avere valori nulli. Anche in questo caso **Direttore** deve essere **UNIQUE** (non possiamo metterlo come chiave primaria perché è opzionale).



Impiegato (Codice, Cognome, Stipendio)

Dipartimento (Nome, Sede, Telefono)

Direzione(Impiegato, Dipartimento, DataInizio)

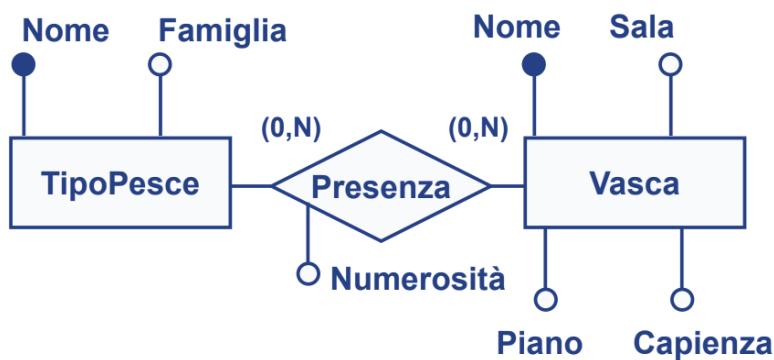
In questo caso, abbiamo un'altra possibilità, quella di andare contro la regola generale e [creare una nuova tabella per l'associazione](#). **Direzione** avrà come chiave **Dipartimento**, e **Impiegato** rimane un semplice attributo, questo però potrebbe portare al fatto che gli impiegati potrebbe dirigere più dipartimenti. Se invece avessimo scelto **Impiegato** come chiave, avremmo che più impiegati potrebbero gestire lo stesso dipartimento.

Quindi se scegliessi questa soluzione dovrai mettere **Dipartimento** o **Impiegato UNIQUE**.

ESERCIZI:

Scrivi lo schema relazionale e i vincoli relazionali.

1.



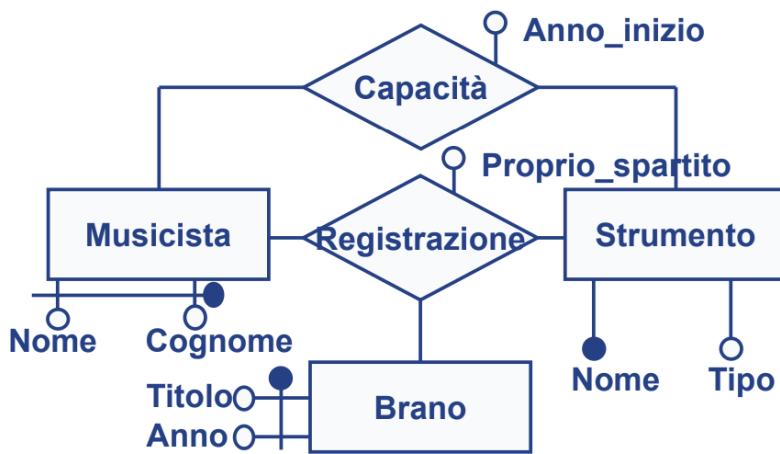
Dato che abbiamo una relazione **molti a molti**, dobbiamo creare la tabella per l'associazione Presenza, con le chiavi delle due entità a cui è collegata, quindi Nome del TipoPesce e Nome della Vasca, per evitare confusione è meglio cambiare i nomi e soprattutto renderli più espressivi, potremmo quindi scegliere TipoPesce e Vasca.

- **TipoPesce** (Nome, Famiglia);
- **Vasca** (Nome, Sala, Piano, Capienza);
- **Presenza** (TipoPesce, Vasca, Numerosità);

Vincoli di integrità referenziale:

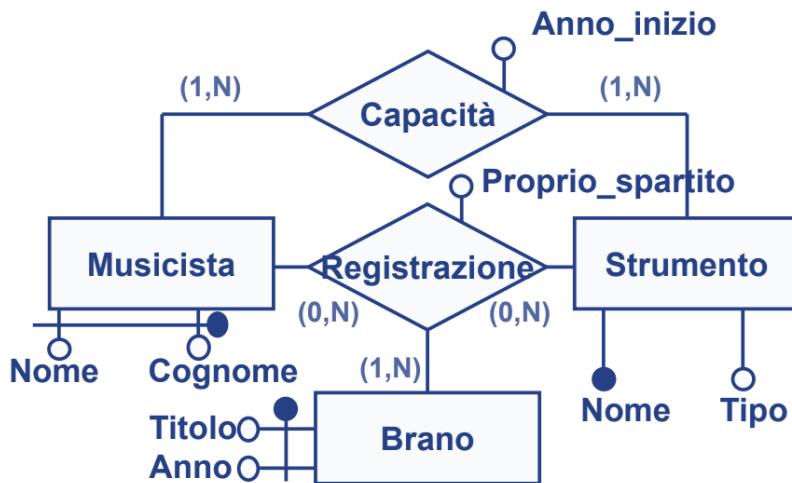
- da **TipoPesce** di Presenza e la **relazione TipoPesce**;
- da **Vasca** di Presenza e la **relazione Vasca**.

2.



In questo esercizio prima dobbiamo mettere i **vincoli di cardinalità**:

- Da musicista a Capacità è **(1,N)**;
- Da Strumento a Capacità è **(1,N)**;
- Da Musicista a Registrazione è **(0,N)**, magari dei musicisti che non registrano nessun brano;
- Da Brano a Registrazione è **(1,N)**, un brano per essere tale deve essere registrato almeno una volta, ma può essere registrato da più musicisti con più strumenti.
- Da Strumento a Registrazione è **(0,N)**, strumenti che non vengono utilizzati in alcuna registrazione.

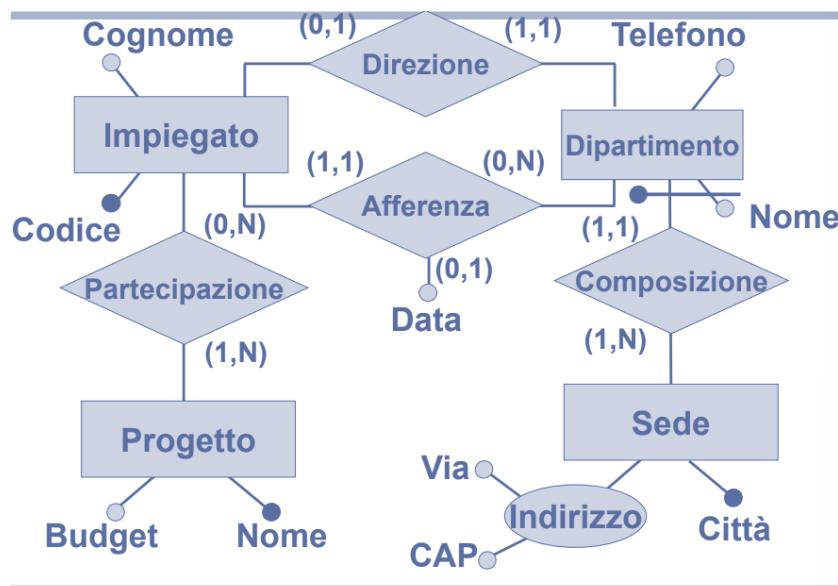


Abbiamo quindi due associazioni molti a molti, quindi oltre le tabelle di Musicista, Strumento e Brano avremo le tabelle di Capacità e Registrazione.

- Musicista(Nome, Cognome);
- Strumento(Nome, Tipo);
- Brano(Titolo, Anno);
- Capacità(NomeMus, CognomeMus, NomeStrum, Anno_inizio, Anno_inizio);
- Registrazione(NomeMus, CognomeMus, NomeStrm, TitoloBrono, AnnoBrono, ProprioStrm).

Ovviamente poi dobbiamo definire i vincoli di integrità referenziale, nel caso di NomeMus e CognomeMus non dobbiamo considerarli come due vincoli separati con Nome e Cognome del Musicista, ma come un unico vincolo che considera la coppia.

3.



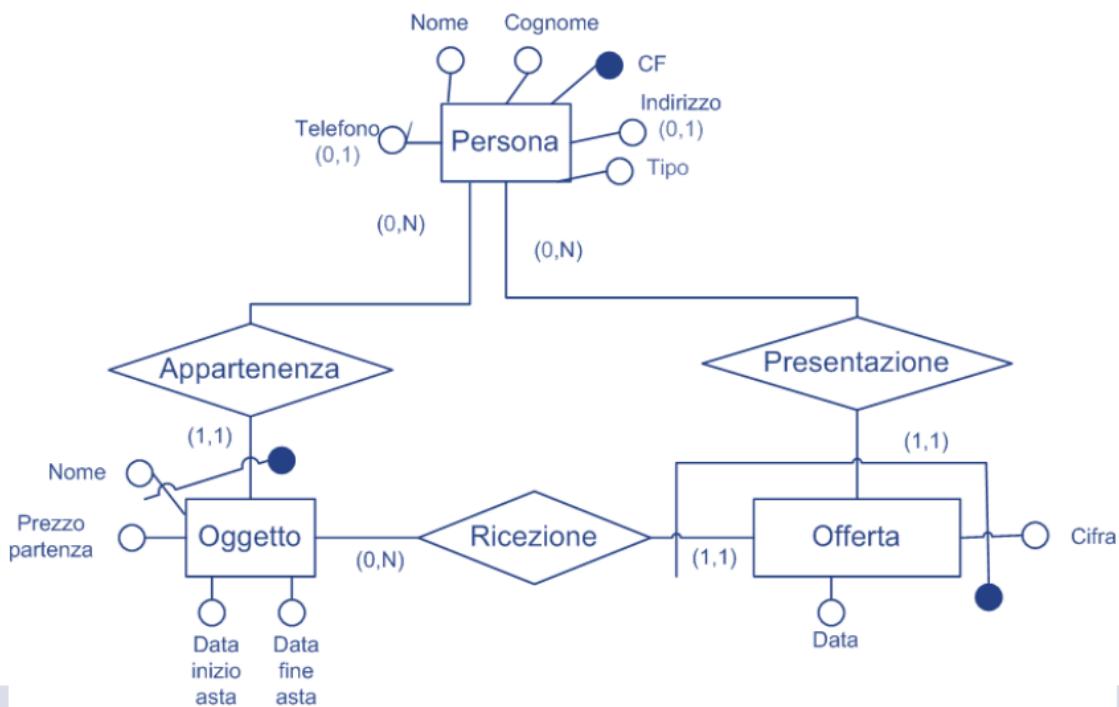
In questo esercizio abbiamo:

- Direzione che è un'associazione **(1,1)**, quindi possiamo decidere di accorpare tutto in una delle due tabelle Impiegato o Dipartimento;
- Afferenza che è un'associazione **(1,N)**, quindi inseriamo gli attributi dell'associazione nella tabella con cardinalità **(1,1)** quindi Impiegato.
- Partecipazione che è una associazione **(N,N)** quindi dobbiamo creare la tabella apposita;
- Composizione che è **(1,N)**, quindi come prima mettiamo tutto nella tabella Dipartimento che è il lato **(1,1)**.

- Sede (Città, Via, CAP);
- Dipartimento (Nome, CittàSede, Telefono, Direttore);
- Impiegato (Codice, Cognome, NomeDip, CittàDip, Data*);
- Progetto (Nome, Budget);
- Partecipazione (Impiegato, Progetto).

Mettiamo Data come opzionale perché l'associazione Afferenza da Dipartimento a Impiegato è (0,N) quindi è possibile che in un Dipartimento non afferisca nessun Impiegato; quindi, la data da cui l'impiegato afferisce deve essere opzionale.

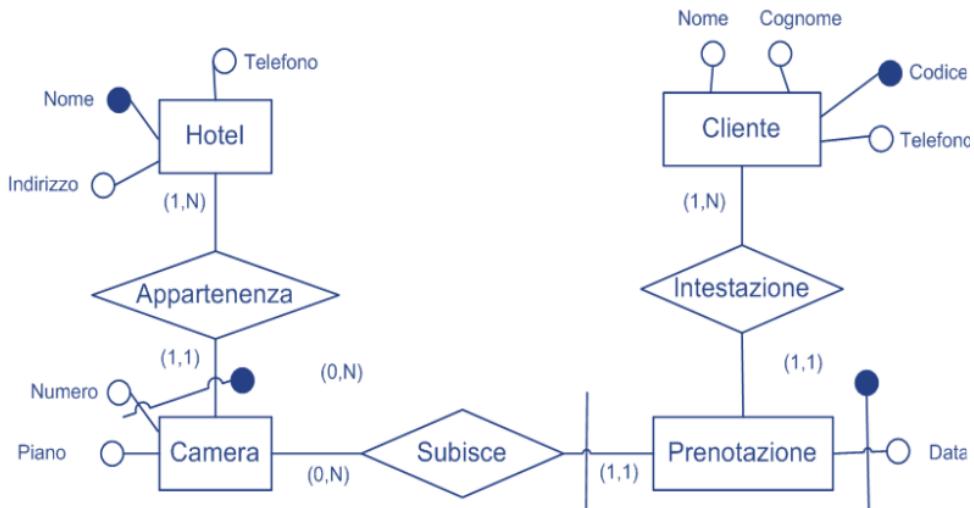
4.



In questo esercizio non abbiamo nessuna associazione molti a molti; quindi, non dobbiamo trasformare le associazioni in tabelle.

- Persona(CF, Nome, Cognome, Tipo, Indirizzo*, Telefono*);
- Oggetto(Nome, CF_Venditore, Prezzo_part, Data_inizio, Data_fine);
- Offerta(Nome_Oggetto, CF_Venditore, CF_Offerente, Cifra, Data);

5.



Anche in questo caso non abbiamo nessuna associazione molti a molti, quindi creiamo solamente le tabelle corrispondenti alle entità.

- Hotel(Nome, Indirizzo, Tel);
- Camera(Numero, NomeHotel, Piano);
- Cliente(Cod, Nome, Cognome, Tel);
- Prenotazione(Data, NumCamera, NomeHotel, CodCliente);

Vincoli di integrità referenziale:

- tra <NumCamera, NomeHotel> di Prenotazione e la relazione Camera;
- tra CodCliente di Prenotazione e la relazione Cliente;
- ecc..

NORMALIZZAZIONE

Alcune "grandi" relazioni presentano anomalie, per questo motivo devono essere decomposte in base alle proprietà dei dati ("dipendenze funzionali"). Le relazioni decomposte soddisfano le forme normali.

Queste anomalie possono derivare dal fatto che ad esempio abbiamo messo dei dati in una tabella, ma quest'ultimi si dovevano inserire in due tabelle diverse.

Le **dipendenze funzionali** sono:

- Boyce-Codd Normal Form;
- Terza forma normale.

Esistono anche le **dipendenze multivalore** (che sono un'estensione di quelle funzionali) che permettono di definire la quarta forma normale.

Una relazione con anomalie

<u>Impiegato</u>	<u>Stipendio</u>	<u>Progetto</u>	<u>Bilancio</u>	<u>Funzione</u>
Rossi	20	Marte	2	tecnico
Verdi	35	Giove	15	progettista
Verdi	35	Venere	15	progettista
Neri	55	Venere	15	direttore
Neri	55	Giove	15	consulente
Neri	55	Marte	2	consulente
Mori	48	Marte	2	direttore
Mori	48	Venere	15	progettista
Bianchi	48	Venere	15	progettista
Bianchi	48	Giove	15	direttore

In questo esempio abbiamo una tabella con 5 colonne, la cui chiave primaria è composta dall'Impiegato e dal Progetto. Non vi sono chiare inconsistenze perché i dati sono corretti, ad esempio allo stesso impiegato corrisponde sempre quello stipendio.

Però lo stipendio di ciascun impiegato è ripetuto in tutte le tuple relative (e anche il bilancio) questo porta a una **ridondanza**.

Se lo stipendio di un impiegato varia, è necessario andarne a modificare il valore in diverse tuple, questo crea un'**anomalia di aggiornamento**.

Se un impiegato interrompe la partecipazione a tutti i progetti, viene cancellato, questo porta ad un'**anomalia di cancellazione**.

Un nuovo impiegato senza progetto non può essere inserito (non possiamo usare valori nulli, perché Progetto è un attributo della chiave), questo porta ad un'**anomalia di inserimento**.

Perché si sono verificati questi fenomeni indesiderabili? Perché abbiamo usato un'unica relazione per rappresentare informazioni eterogenee:

- Gli impiegati con i relativi stipendi;
- I progetti con i relativi bilanci;
- Le partecipazioni degli impiegati ai progetti con le relative funzioni.

La normalizzazione è una procedura che viene svolta dopo la progettazione logica, Serve a identificare possibili anomalie e correggerle, decomponendo le relazioni "scorrette" in più relazioni.

Una **forma normale** è una proprietà di una base di dati relazionale che ne garantisce la "qualità". Quando una relazione non è normalizzata:

- Presenta ridondanze;
- Si presta a comportamenti poco desiderabili durante gli aggiornamenti.

Le forme normali sono utili anche per:

- Memorizzazione efficiente su DBMS;
- Ottimizzazione della risposta alle query.

La normalizzazione è quindi una procedura che permette di trasformare schemi non normalizzati in schemi che soddisfano una **forma normale**, va utilizzata come tecnica di verifica dei risultati della progettazione di una base di dati. Non costituisce una metodologia di progettazione.

Per studiare in maniera sistematica questi aspetti, è necessario introdurre un vincolo di integrità: **la dipendenza funzionale**, che ci dice che un certo dato dipende funzionalmente da altri dati. Ad esempio, dato in input un impiegato ci restituisce in output un unico valore che è il suo stipendio.

Esempi di dipendenze funzionali:

- Ogni impiegato ha un solo stipendio (anche se partecipa a più progetti);
- Ogni progetto ha un solo bilancio;
- Ogni impiegato in ciascun progetto ha una sola funzione (anche se può avere funzioni diverse in progetti diversi).

Consideriamo una relazione **R** con attributi **X**, e due sottoinsiemi non vuoti **Y** e **Z** di **X**. Esiste in **R** una **dipendenza funzionale** (FD) da **Y** (ad esempio Impiegato) a **Z** (ad esempio Stipendio) se, per ogni coppia di tuple **t1** e **t2** di **R** con gli stessi valori su **Y**, risulta che **t1** e **t2** hanno gli stessi valori anche su **Z**.

La notazione è: **Y → Z**, e si legge **Z** dipende funzionalmente da **Y**, oppure **Y** determina **Z**.

Esempi:

- Impiegato → Stipendio;
- Progetto → Bilancio;
- Impiegato, Progetto → Funzione.

Le FD sono basate sulla conoscenza del dominio dei dati che si stanno modellando ("sul mondo reale"), le FD casuali (trovate nei dati) che non hanno corrispondenza col mondo reale non vanno considerate.

Alcune dipendenze funzionali vengono chiamate **banali**, ad esempio **Impiegato, Progetto → Progetto**, perché la **parte a destra è sottoinsieme della parte sinistra**, questa dipendenza è sempre vera, quindi sono sempre soddisfatte per definizione.

Un altro caso è quando l'insieme di attributi in input sono chiave, in quel caso la funzione mi restituisce ogni altro attributo della tabella.

Y → Z è non banale se nessun attributo in **Z** appartiene a **Y**, e se **Y** non è superchiave. Se **Y** è (super)chiave (come nella seconda FD) allora **Y → Z vale per qualunque Z**.

Immaginiamo di avere una tabella R con tre attributi A,B,C e abbiamo una dipendenza funzionale da $A \rightarrow B$:

A	B	C
1	2	c1
1	2	c2

In questo caso ad $A = 1$ corrisponde $B = 2$, quindi alla prossima istanza di $A = 1$ corrisponderà sempre quel valore di B.

Consideriamo ora una tabella R con attributi A,B,C,D e con dipendenza funzionale $A \rightarrow B$ e $B \rightarrow C$. Allora [possiamo derivare anche \$A \rightarrow C\$](#) :

A	B	C	D
1	2	3	d1
1	2	3	d2

In questo caso $A = 1$ implica $B = 2$, e $B = 2$ implica $C = 3$, questo ci dice anche che $A = 1$ implica $C = 3$, questo perché vale la [proprietà transitiva](#).

Supponiamo ora che A determini tutti gli altri attributi di R: R (A, B, C), $A \rightarrow B, C$

A	B	C
1	2	3
1	2	3

Se è così, non posso avere 2 tuple diverse con valori di A uguali (altrimenti anche il resto delle 2 tuple sarebbe uguale), quindi [A è superchiave](#).

Esercizio

Considerare la seguente relazione e individuare le chiavi e le dipendenze funzionali. Individuare inoltre eventuali ridondanze e anomalie nella relazione.

Docente	Dipartimento	Facoltà	Preside	Corso
Verdi	Matematica	Ingegneria	Neri	Analisi
Verdi	Matematica	Ingegneria	Neri	Geometria
Rossi	Fisica	Ingegneria	Neri	Analisi
Rossi	Fisica	Scienze	Bruni	Analisi
Bruni	Fisica	Scienze	Bruni	Fisica

Partiamo trovando le dipendenze funzionali:

- Docente \rightarrow Dipartimento, questo vuol dire che un docente afferisce al massimo ad un dipartimento;
- Facoltà \rightarrow Preside, una facoltà ha un unico preside. Questa però introduce una **ridondanza** perché per ogni Corso nella stessa Facoltà, il Preside deve essere ripetuto.

Inoltre, abbiamo un'anomalia di aggiornamento perché se cambiemo il preside dobbiamo aggiornare tutte le tuple che lo contengono.

La chiave potrebbe essere l'insieme di **Docente, Facoltà e Corso**.

REGOLE PER FD:

Regola di Splitting:

Se $A \rightarrow B_1, B_2, \dots, B_n$

Allora:

$A \rightarrow B_1,$

$A \rightarrow B_2,$

...

$A \rightarrow B_n$

Possiamo fare splitting sul lato sinistro? Spoiler, no.

Se $A_1, A_2, \dots, A_n \rightarrow B$

Allora:

$A_1 \rightarrow B,$

$A_2 \rightarrow B,$

...

$A_n \rightarrow B$

Regola di combinazione:

Se:

$A \rightarrow B_1,$

$A \rightarrow B_2,$

...

$A \rightarrow B_n$

Allora:

$A \rightarrow B_1, B_2, \dots, B_n$

Proprietà simmetrica:

Se:

$A \rightarrow B$ allora: $B \rightarrow A$

NO!!

Non vale la proprietà simmetrica.

Dati uno schema di relazione $R(U)$, un insieme F di FD definite su U e un insieme di attributi X contenuti in U (cioè $X \subseteq U$). La **chiusura di X rispetto ad F** , indicata con X^+_F , è l'insieme degli attributi che dipendono funzionalmente da X :

$$X^+_F = \{ A \mid A \in U \text{ e } F \text{ implica } X \rightarrow A \}$$

In altri termini se A appartiene a X^+_F allora $X \rightarrow A$ è implicata da F .

ESEMPIO:

Iscrizione a un test di ammissione:

Iscrizione (CF, Nome, Indirizzo, VotoM, PuntPartenza, CodCorsoStudi, Univ, CS).

Lo studente con **codice fiscale CF**, di cui si conosce **Nome**, **Indirizzo**, **voto di maturità VotoM**, si iscrive al test di ammissione del corso di studi **CS** dell'università **Univ**, identificato dal codice **CodCorsoStudi**, e il suo punteggio di partenza è **PuntPartenza** (calcolato in base al voto di maturità **VotoM**)

Lo stesso studente si può iscrivere al test di più corsi di studio; quindi, la chiave di Iscrizione è **<CF, CodCorsoStudi>**.

Quali sono le dipendenze funzionali? Vogliamo specificare: l'insieme minimale F di dipendenze funzionali non banali **tale che tutte le FD valide sulla relazione derivino da F** .

- $CF \rightarrow \text{Nome, Indirizzo, VotoM};$
- $\text{VotoM} \rightarrow \text{PuntPartenza};$
- $\text{CodCorsoStudi} \rightarrow \text{Univ, CS}$

Qual è la chiusura di $X = \{ CF, \text{CodCorsoStudi} \}$? Visto che sono chiave la chiusura sarà tutti i valori della tabella.

Però per sicurezza andiamo a verificare:

- Passo 0: $X^+_F = X = \{ CF, \text{CodCorsoStudi} \};$
- Passo 1 aggiungo le dipendenze da CF : $X^+_F = \{ CF, \text{CodCorsoStudi}, \text{Nome, Indirizzo, VotoM} \};$
- Passo 2 aggiungo le dipendenze da CodCorsoStudi : $X^+_F = \{ CF, \text{CodCorsoStudi}, \text{Nome, Indirizzo, VotoM, Univ, CS} \};$
- Passo 3, posso usare la dipendenza rimasta ($\text{VotoM} \rightarrow \text{PuntPartenza}$)? Sì, perchè VotoM appartiene a X^+_F :

$$X^+_F = \{ CF, \text{CodCorsoStudi}, \text{Nome, Indirizzo, VotoM, Univ, CS, PuntPartenza} \}.$$

Ottengo tutte gli attributi della relazione e così capisco che X è superchiave.

In generale:

- In Input ho un insieme X di attributi e un insieme F di dipendenze funzionali;
- In Output avrò un insieme X_P di attributi;
 1. Inizializziamo X_P con l'insieme di input X ;
 2. Se esiste una FD, da $Y \rightarrow A$ in F con $Y \subseteq X_P$ e $A \notin X_P$, allora aggiungiamo A (la parte destra) a X_P ;
 3. Ripetiamo il passo (2) fino a quando non ci sono ulteriori attributi che possono essere aggiunti a X_P .

Un insieme di attributi K è **(super)chiave** per uno schema di relazione $R(U)$ su cui è definito un insieme di dipendenze funzionali F , se F implica $K \rightarrow U$.

Scritto diversamente: se $K^+_F = U$.

Quindi, per trovare le chiavi posso procedere per tentativi, partendo dai singoli attributi, passando poi alle coppie, ecc., finché non trovo una chiusura che comprende tutti gli attributi.

ESERCIZIO:

Data la relazione R(A, B, C, D, E, F, G, H, I) e le dipendenze funzionali FD:

- A → C, D;
- C → I;
- D → F;
- E → H;
- F → E, B.

L'attributo A è chiave di R?

Dobbiamo calcolare la chiusura di A, e se comprende tutti gli attributi allora è chiave.

- Passo 0: $A^+ = \{A\}$;
- Passo 1: $A^+ = \{A, \textcolor{blue}{C}, \textcolor{blue}{D}\}$;
- Passo 2: $A^+ = \{A, C, D, \textcolor{blue}{I}\}$;
- Passo 3: $A^+ = \{A, C, D, I, \textcolor{blue}{F}\}$;
- Passo 4: $A^+ = \{A, C, D, I, F, \textcolor{blue}{E}, \textcolor{blue}{B}\}$;
- Passo 5: $A^+ = \{A, C, D, I, F, E, B, \textcolor{blue}{H}\}$;

A non è chiave perché nella chiusura manca la G.

Che altra dipendenza funzionale potrei avere perché A sia chiave di R? Ad esempio, B → G.

Qual è una (super)chiave di R? In questo caso sarebbe $\langle A, G \rangle$. Ma è minimale? Sì, perché $G_{+ FD} = \{G\}$, quindi G non è chiave (e sappiamo già che A non è chiave).

Le anomalie sono legate ad alcune FD:

1. Gli impiegati hanno un unico stipendio, Impiegato → Stipendio;
2. I progetti hanno un unico bilancio, Progetto → Bilancio;
3. In ciascun progetto, un impiegato svolge una sola funzione, Impiegato, Progetto → Funzione

Nella terza il soddisfacimento è più "semplice", perché Impiegato, Progetto è chiave. Le prime due FD non corrispondono a chiavi e causano anomalie.

La relazione contiene alcune dipendenze legate alla chiave e altre ad attributi che non formano una chiave. Le **anomalie** sono causate dalla presenza di concetti eterogenei:

- Proprietà degli impiegati (lo stipendio);
- Proprietà di progetti (il bilancio);
- Proprietà della chiave Impiegato, Progetto.

Una relazione R è in **forma normale di BoyceCodd** (BCNF) se, per ogni dipendenza funzionale $X \rightarrow Y$ definita su di essa, X è (super)chiave di R.

La forma normale richiede che i concetti in una relazione siano **omogenei** (solo proprietà direttamente associate alla chiave).

Che facciamo se una relazione non soddisfa la BCNF? La rimpiazziamo con altre relazioni che soddisfano la BCNF. Come? **Decomponendo sulla base delle dipendenze funzionali**, al fine di separare i concetti.

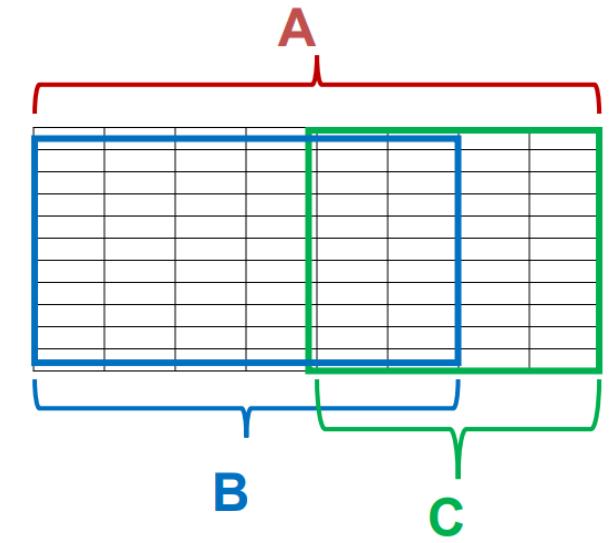
Cosa vuol dire decomporre una relazione? Parto da una relazione grande R(A) dove A = A₁, ..., A_n.

Decompongo R in 2 relazioni R₁ e R₂:

- R₁(B) dove B = B₁, ..., B_m;
- R₂(C) dove C = C₁, ..., C_k.

Le proprietà richieste sono:

- $B \cup C = A$ (conservo tutti gli attributi di R);
- $R1 \text{ NAT_JOIN } R2 = R$ (facendo il join ri-ottengo R).



La [procedura intuitiva di normalizzazione](#) procede in questo modo:

- Per ogni dipendenza $X \rightarrow Y$ che viola la BCNF, definisco una relazione su XY ed elimino Y dalla relazione originaria.

Questa procedura non è valida in generale, ma solo nei “casi semplici”.

(1) Impiegato → Stipendio

(2) Progetto → Bilancio

<u>Impiegato</u>	<u>Stipendio</u>	<u>Progetto</u>	<u>Bilancio</u>	<u>Funzione</u>
Rossi	20	Marte	2	tecnico
Verdi	35	Giove	15	progettista
Verdi	35	Venere	15	progettista
Neri	55	Venere	15	direttore
Neri	55	Giove	15	consulente
Neri	55	Marte	2	consulente
Mori	48	Marte	2	direttore
Mori	48	Venere	15	progettista
Bianchi	48	Venere	15	progettista
Bianchi	48	Giove	15	direttore

(1) e (2) violano la BCNF

Impiegato viola la BCNF perché Impiegato da solo non è chiave, e anche Progetto la viola per lo stesso motivo. Funzione non ha FD non banali, quindi non viola BCNF.

Quindi in questo caso la soluzione è lasciare gli attributi Impiegato, Funzione e Progetto nella relazione originale, e creare due nuove relazioni con Impiegato e Progetto (con Impiegato come chiave) e una con Progetto e Bilancio (con chiave Progetto).

Impiegato	Progetto	Funzione
Rossi	Marte	tecnico
Verdi	Giove	progettista
Verdi	Venere	progettista
Neri	Venere	direttore
Neri	Giove	consulente
Neri	Marte	consulente
Mori	Marte	direttore
Mori	Venere	progettista
Bianchi	Venere	progettista
Bianchi	Giove	direttore

Impiegato	Stipendio
Rossi	20
Verdi	35
Neri	55
Mori	48
Bianchi	48

Progetto	Bilancio
Marte	2
Venere	15
Giove	15

Ricomponendo (tramite join) la relazione originale, ottengo le stesse tuple di partenza. (NAT_JOIN Impiegati NAT_JOIN Progetti).

Tramite questa decomposizione non si viola la BCNF perché:

- Impiegato → Stipendio, e Impiegato è chiave;
- Progetto → Bilancio, e Progetto è chiave;
- Impiegato, Progetto → Funzione, e Impiegato e Progetto sono chiavi.

R

Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

Impiegato → Sede
Progetto → Sede

In questo caso, abbiamo che Sede dipende sia da Impiegato che da Progetto, e viola la BCNF perché le parti sinistre delle dipendenze sono chiavi. Utilizzando la tecnica di prima dovremmo scomporre la relazione originale in due relazioni una con Impiegato e Sede, e una con Progetto e Sede.

delle dipendenze

R

Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

R1

Impiegato	Sede
Rossi	Roma
Verdi	Milano
Neri	Milano

R2

Progetto	Sede
Marte	Roma
Giove	Milano
Saturno	Milano
Venere	Milano

Provando a ricostruire R con R1 NAT_JOIN R2, otteniamo una relazione che è diversa da quella di partenza.

Impiegato	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano
Verdi	Saturno	Milano
Neri	Giove	Milano

Se avessimo lasciato la relazione originale con la coppia Impiegato e Progetto:

Impiegato	Sede	Progetto	Sede
Rossi	Roma	Marte	Roma
Verdi	Milano	Giove	Milano
Neri	Milano	Saturno	Milano
		Venere	Milano

Impiegato	Progetto
Rossi	Marte
Verdi	Giove
Verdi	Venere
Neri	Saturno
Neri	Venere

Non va bene perché si perdono informazioni, se ho un impiegato di Roma che lavora ad un progetto di Milano, qual è la sede? Questo caso, quindi, **viola il vincolo della decomposizione senza perdita**.

Una relazione R si **decomponte senza perdita** su R1 e R2 se il join naturale di R1 e R2 è uguale a R (cioè il join non contiene tuple spurie e non ci sono tuple mancanti).

La decomposizione senza perdita è garantita se gli attributi comuni contengono una chiave per almeno una delle relazioni decomposte. Nell'esempio di prima, Sede non è chiave né di R1 né di R2.

Algoritmo di decomposizione BCNF:

- In Input ho una relazione R e le sue dip. Funzionali;
- In Output avrò una decomposizione di R in BCNF;
 - Ripetere fin quando tutte le relazioni sono in BCNF:
 1. Scegliere R' che viola BCNF;
 2. Decomporla in R1(A,B) e R2(A,C) in modo che gli attributi comuni (A) siano chiave di R1 o di R2 (o di entrambe);
 3. Ricalcolare FD.

Esercizio:

Consideriamo l'esercizio di prima: Iscrizione (CF, Nome, Indirizzo, VotoM, PuntPartenza, CodCorsoStudi, Univ, CS).

Con queste dipendenze funzionali?

- CF → Nome, Indirizzo, VotoM;
- VotoM → PuntPartenza;
- CodCorsoStudi → Univ, CS

Decomponiamo Iscrizione in modo che non venga violata la BCNF:

- Passo 0: Iscrizione (CF, Nome, Indirizzo, VotoM, PuntPartenza, CodCorsoStudi, Univ, CS), considero la prima dipendenza funzionale VotoM → PuntPartenza;
- Passo 1: scompongo la prima dip. Funzionale che viola la BCNF perché VotoM non è chiave, e rimuovo PuntPartenza da Iscrizione:
 - Punteggio (VotoM, PuntPartenza);
 - Iscrizione (CF, Nome, Indirizzo, VotoM, CodCorsoStudi, Univ, CS);
- Passo 2: abbiamo la seconda dip. Funzionale, CF → Nome, Indirizzo, VotoM, che viola la BCNF perché CF da solo non è chiave:
 - Iscritto (CF, Nome, Indirizzo, VotoM);
 - Punteggio (VotoM, PuntPartenza);
 - Iscrizione (CF, CodCorsoStudi, Univ, CS).
- Passo 3: abbiamo la terza dip. Funzionale, CodCorsoStudi → Univ, CS, anche qui viola la BCNF:
 - Corsi (CodCorsoStudi, Univ, CS);
 - Iscritto (CF, Nome, Indirizzo, VotoM);
 - Punteggio (VotoM, PuntPartenza);
 - Iscrizione (CF, CodCorsoStudi).

In genere la **BCNF** fornisce "buoni" schemi, a volte però non è raggiungibile. Per raggiungerla si perderebbero alcune informazioni, che non sono solo valori ma anche delle dipendenze funzionali.

Si può usare un suo rilassamento: la **3NF**, che non protegge da tutte le anomalie. Per questo motivo esiste anche una forma normale più restrittiva: la **4NF**.

<u>Impiegato</u>	<u>Progetto</u>	<u>Sede</u>
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Venere	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

\

Abbiamo perso una FD,
cioè una informazione
potenzialmente utile

<u>Impiegato</u>	<u>Sede</u>
Rossi	Roma
Verdi	Milano
Neri	Milano

<u>Impiegato</u>	<u>Progetto</u>
Rossi	Marte
Verdi	Giove
Verdi	Venere
Neri	Saturno
Neri	Venere

Impiegato → Sede

Progetto → Sede

In questo esempio di prima, utilizzando la BCNF abbiamo perso una dipendenza funzionale, che poteva essere un'informazione potenzialmente utile.

Una **decomposizione conserva le dipendenze** se ciascuna delle dipendenze funzionali dello schema originario coinvolge attributi che compaiono tutti insieme in uno degli schemi decomposti (Progetto → Sede non è conservata).

Una decomposizione dovrebbe sempre soddisfare:

- La **decomposizione senza perduto**, che garantisce la ricostruzione delle informazioni originarie;
- La **conservazione delle dipendenze**, che garantisce il mantenimento dei vincoli di integrità originari.

Consideriamo ora questa relazione non normalizzata:

Dirigente	Progetto	Sede
Rossi	Marte	Roma
Verdi	Giove	Milano
Verdi	Marte	Milano
Neri	Saturno	Milano
Neri	Venere	Milano

Progetto, Sede → Dirigente Dirigente → Sede

La decomposizione risulta problematica perché Progetto, Sede → Dirigente, coinvolge tutti gli attributi e quindi nessuna decomposizione può preservare tale dipendenza. Nel caso in cui provassimo a svolgere la decomposizione in BCNF, dovremmo creare una tabella con Dirigente e Sede, con Dirigente come chiave, e Sede bisognerà rimuoverlo dalla tabella originaria che diventerà solo Progetto e Dirigente, con chiave Progetto. In questo caso stiamo proprio modificando la semantica dello schema.

Quindi in alcuni casi la BCNF non è raggiungibile.

Esercizi:

- ↙ Esami (CodEsame, Nome, CodProf, NomeProf, NumStudenti, Dip)

Per cui valgono queste le dipendenze funzionali:

- CodEsame → Nome, NumStudenti;
- CodProf → Dip, NomeProf.

Portare Esami in BCNF, eseguendo una suddivisione senza perdite e che mantenga le dipendenze funzionali.

Non rispetta la BCNF perché né CodEsame né CodProf da soli sono chiavi, e non è neanche in 3NF perché né Nome, né NumStudenti, né Dip e né NomeProf appartengano ad una chiave.

Creiamo una relazione per ciascuna FD, e rimuoviamo dalla tabella originaria le parti destre:

- Corso (CodEsame, Nome, NumStudenti);
- Prof (CodProf, Dip, NomeProf);
- Esami (CodEsame, CodProf).

Lo schema è in 3NF e in BCNF perché la parte sinistra di ogni FD è chiave.

-
- ↙ Individuare le FD presenti in CorsiStud, che contiene informazioni sui corsi seguiti dagli studenti:

CorsiStud (CodCorso, NomeCorso, NomeProf, CodProf, Dip, MatricolaStud, NomeStud, NumeroOreCorso, NumeroCreditiCorso)

Si supponga che:

- Ciascun corso sia tenuto da un solo docente;
- Ciascun professore afferisce ad un solo dipartimento;
- Ciascun professore possa tenere più corsi.

Le dipendenze funzionali sono:

- Per la prima abbiamo CodCorso → CodProf, e all'interno metteremo anche tutte le informazioni dirette sul corso;
- Per la seconda abbiamo CodProf → Dip, e all'interno metteremo anche il NomeProf;
- La terza è già implicita in CodCorso → CodProf;
- Manca solo MatricolaStud che determina NomeStud.

In conclusione quindi abbiamo:

- CodCorso → NomeCorso, NumeroOreCorso, NumCreditiCorso, CodProf;
- CodProf → NomeProf, Dip;
- MatricolaStud → NomeStud.

La relazione CorsiStud è in 3NF? È in BCNF?

Non è in 3NF perché, considerando ad esempio la terza FD, MatricolaStud non è superchiave, e NomeStud non appartiene a nessuna chiave. Non essendo in 3NF non è nemmeno in BCNF.

Decomporre CorsiStud in BCNF:

- Corsi (CodCorso, NomeCorso, NumeroOreCorso, NumCreditiCorso, CodProf);
- Prof (CodProf, NomeProf, Dip);
- Stud (MatricolaStud, NomeStud);
- Frequenze (CodCorso, MatricolaStud).

-
- Considerare la relazione R(A, B, C, D, E, F, G), con queste dipendenze funzionali:
 $FD = \{C \rightarrow AB, BC \rightarrow DE, D \rightarrow B, F \rightarrow C\}$

Calcolare la chiusura di {A,B}:

$$\{A, B\}^+ = \{A, B\};$$

Calcolare la chiusura di { C }:

- Considero la prima dipendenza $C \rightarrow AB$: $\{C\}^+ = \{C, A, B\}$;
- Aggiungo la dipendenza $BC \rightarrow DE$: $\{C\}^+ = \{A, B, C, D, E\}$.

Dimostrare che $\langle F, G \rangle$ è superchiave di R:

Calcolo la chiusura di { F, G }:

- Considero per prima $F \rightarrow C$: $\{F, G\}^+ = \{F, G, C\}$;
- Considero poi $C \rightarrow AB$: $\{F, G\}^+ = \{F, G, C, A, B\}$;
- Considero $BC \rightarrow DE$: $\{F, G\}^+ = \{F, G, C, A, B, D, E\}$;
- Ottengo $\{F, G\}^+ = \{A, B, C, D, E, F, G\}$, quindi ho tutti gli attributi di R, quindi è superchiave di R.

- Considerare la relazione R(A, B, C, D, E, F, G, H), con queste dipendenze funzionali:
 $FD = \{AB \rightarrow C, CD \rightarrow FG, F \rightarrow E, E \rightarrow AH, H \rightarrow D\}$

Esiste una dipendenza funzionale tra E e D? Dimostrarlo.

Calcolo la chiusura di { E }:

$$\{E\}^+ = \{A, D, E, H\} \text{ comprende } D, \text{ quindi } E \rightarrow D.$$

Esiste una dipendenza funzionale tra EF e AG? Dimostrarlo.

Calcolo la chiusura di { EF }:

$$\{E, F\}^+ = \{A, D, E, F, H\} \text{ comprende } A \text{ ma non } G; \text{ quindi, } EF \text{ non determina } AG.$$

DBMS RELAZIONALI: CARATTERISTICHE EVOLUTE

Una **transazione** è una sequenza di operazioni da considerare indivisibile ("atomica"), corretta anche in presenza di concorrenza e con effetti definitivi.

Un esempio di transazione è un bonifico di 100 euro dal conto di Tim al conto di Tom:

- Inizio_transazione;
- Sottrai 100 euro dal saldo del conto di Tim;
- Aggiungi 100 euro al saldo del conto di Tom;
- Fine transazione.

Le transazioni possiedono le proprietà **ACID** ("acide"):

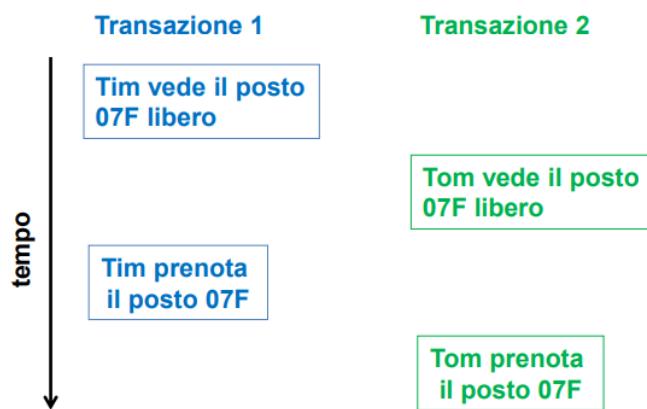
- **Atomicità**;
- **Consistenza**;
- **Isolamento**;
- **Durabilità (persistenza)**.

L'**atomicità** garantisce che la sequenza di operazioni sulla base di dati viene eseguita per intero o per niente. Nel esempio del trasferimento di fondi da un conto A ad un conto B, o si fa il prelevamento da A e il versamento su B o nessuno dei due.

La **consistenza** invece garantisce che al termine dell'esecuzione di una transazione, i vincoli di integrità sono soddisfatti. "Durante" l'esecuzione ci possono essere violazioni, ma se restano alla fine allora la transazione deve essere annullata per intero.

L'**isolamento** garantisce che l'effetto di transazioni concorrenti deve essere coerente (equivalente all'esecuzione seriale). Se due assegni emessi sullo stesso conto corrente vengono incassati contemporaneamente si deve evitare di trascurarne uno.

Se viene violato l'isolamento le transazioni vengono eseguite in parallelo e questo comporta delle anomalie.



La **durabilità** consiste nel fatto che la conclusione positiva di una transazione corrisponde ad un impegno (**commit**), che serve per mantenere traccia del risultato in modo definitivo, anche in presenza di guasti. Il commit viene fornito dal sistema solo quando è sicuro che i dati sono stati salvati in modo permanente.

TRANSAZIONI IN SQL

Una **transazione** inizia al primo comando SQL dopo la "connessione" di un utente alla base di dati oppure alla conclusione di una precedente transazione (lo standard indica anche un comando start transaction, non obbligatorio, e quindi non previsto in molti sistemi).

Una transazione si conclude correttamente quando le operazioni specificate a partire dall'inizio della transazione vengono eseguite sulla base di dati, e successivamente viene eseguita la **commit** con la quale si rendono definitive queste operazioni.

Per gestire quei casi in cui qualcosa non è andata come si voleva si utilizza la **rollback**, con la quale si rinuncia all'esecuzione delle operazioni specificate dopo l'inizio della transazione. L'applicazione dice al sistema che la transazione è terminata, e chiede che debba essere annullata, cioè di tornare allo stato iniziale.

Una transazione in SQL

```
start transaction;  
update ContoCorrente  
    set Saldo = Saldo - 10  
    where NumeroConto = 123;  
update ContoCorrente  
    set Saldo = Saldo + 10  
    where NumeroConto = 456;  
commit work;
```

I DBMS relazionali garantiscono le proprietà ACID, ma garantirle non è semplice perché ha un impatto negativo sull'efficienza del DBMS.

Nuovi DBMS non-relazionali (NoSQL) supportano versioni deboli di ACIDità per essere più efficienti, i DBMS relazionali di nuova generazione (NewSQL) cercano di conciliare ACID ed efficienza.

BASI DI DATI ATTIVE

Le **basi di dati attive** sono quelle basi di dati che contengono **regole attive** (chiamate **trigger**). Utilizzano il paradigma Evento-Condizione-Azione, cioè quando un evento si verifica, se la condizione è vera, allora l'azione è eseguita.

- Un **evento** è normalmente una modifica dello stato del database: insert, delete, update). Quando accade l'evento, il trigger è attivato;
- Una **condizione** è un predicato che identifica se l'azione del trigger deve essere eseguita;
- Un'**azione** è una sequenza di comandi SQL.

Questo modello ci consente di programmare il nostro DBMS per fargli eseguire autonomamente delle azioni, quindi permette delle **computazioni reattive**.

Il vantaggio, rispetto ad implementare le regole esternamente è che la **logica dei dati è separata dalla logica delle applicazioni** (se la logica dice che quando devo eliminare un vigile devo rimuovere anche tutte le multe effettuate da lui, allora tutte le applicazioni che operano sui vigili opereranno allo stesso modo).

Un tipo di computazione reattiva sono le **azioni compensative nei vincoli di integrità**, ad esempio, quando definiamo un vincolo di chiave esterna le azioni compensative sono ON DELETE CASCADE.

Ogni trigger è caratterizzato da:

- Nome;
- Target (tabella controllata);
- Modalità (before o after);
- Evento (insert, delete o update);
- Granularità (statement-level o row-level);
- Azione;
- Timestamp di creazione.

La sintassi standard non è supportata da molti DBMS.

Sintassi SQL

```
create trigger NomeTrigger
{before | after }
{ insert | delete | update [of Column] } on
TabellaTarget
[referencing
 {[old table [as] VarTuplaOld]
 [new table [as] VarTuplaNew] } |
 {[old [row] [as] VarTabellaOld]
 [new [row] [as] VarTabellaNew] }]
[for each { row | statement }]
[when Condizione]
StatementProceduraleSQL
```

La granularità degli eventi può essere in:

- **Modalità statement-level** (di default): il trigger viene considerato e possibilmente eseguito solo una volta per ogni statement (comando) che lo ha attivato, indipendentemente dal numero di tuple modificate. È in linea con SQL (set-oriented);
- **Modalità row-level**: il trigger viene considerato e possibilmente eseguito una volta per ogni tupla modificata.

Se granularità = **row-level**, abbiamo due variabili:

- **old**: variabile che fa riferimento alla tupla nello stato precedente la modifica;
- **new**: variabile che fa riferimento alla tupla nello stato prodotto dalla modifica.

Se granularità = **statement-level**, non ho una variabile di riferimento per tupla, ma ho un'unica variabile per la tabella prima della modifica e dopo la modifica.

- **old_table**: stato precedente della porzione di tabella interessata dalla modifica;
- **new_table**: stato prodotto dalla modifica.

Le modalità possono essere:

- **BEFORE**: il trigger è considerato e possibilmente eseguito prima della modifica.
I trigger before non possono modificare lo stato del database; possono al più condizionare i valori "new".
Normalmente questa modalità è usata quando si vuole verificare una modifica prima che essa avvenga e "modificare la modifica".

```
create trigger LimitaAumenti  
before update of Salario on Impiegato  
for each row  
when (New.Salario > Old.Salario * 1.2)  
set New.Salario = Old.Salario * 1.2
```

Un esempio è il **conditioner** che agisce prima dell'update e della verifica di integrità.

In questo esempio creiamo un trigger LimitaAumenti, quest'ultimo viene attivato quando si vorrebbe incrementare il Salario più del 20%, infatti viene eseguito prima ancora di modificare effettivamente il campo.

Tramite questo trigger posso evitare che venga aumentato il salario più del 20%.

- **AFTER**: il trigger è considerato e eseguito dopo l'evento, è la modalità più comune.

```
create trigger LimitaAumenti  
after update of Salario on Impiegato  
for each row  
when (New.Salario > Old.Salario * 1.2)  
update Impiegato  
set New.Salario = Old.Salario * 1.2  
where ImpNum = new.ImpNum
```

Un esempio è il **re-installer** che agisce dopo l'update.

In questo esempio dopo aver effettuato l'update, verifico per ogni riga se il nuovo salario è > di quello vecchio * 1,2, se è così eseguo una nuova update e modifco il salario nuovo.

Sembra più efficiente il before perché fa una solo scrittura, rispetto alle due che effettua l'after.

Esempio di trigger row-level

```
create trigger AccountMonitor  
after update on Account  
for each row  
when new.Totale < old.Totale  
insert values  
(new.NumeroConto,  
old.Totale-new.Totale, now())  
into Prelievi
```

Esempio di trigger statement-level

```
create trigger  
ArchiviaFattureCancellate  
after delete on Fattura  
insert into FattureCancellate  
(select *  
from old_table)
```

Nel esempio di trigger row-level, dopo che si effettua un update si verifica ciascuna riga che è stata modificata, e si verifica la condizione, se è rispettata viene aggiunta una nuova riga nella tabella Prelievi, con quanto si è prelevato e da chi.

Nel esempio di statement level, dopo che si effettua la delete in Fattura si inserisce in un'altra tabella FattureCancellate le fatture che sono state eliminate dall'altra tabella (old_table).

Quando vi sono più trigger associati allo stesso evento (**in conflitto**) vengono eseguiti come segue:

- Per primi i **before** triggers;

- Poi viene eseguita la **modifica**;
- Infine, sono eseguiti gli **after triggers**.

Quando vari trigger appartengono alla stessa categoria, l'ordine di esecuzione è definito in base al loro timestamp di creazione, i **trigger più vecchi hanno priorità più alta**.

L'azione di un trigger può produrre eventi che attivano altri trigger, si possono quindi **generare dei cicli infiniti**.

Possibili esiti:

- L'esecuzione termina correttamente;
- L'esecuzione termina in errore quando si raggiunge una data profondità di ciclo.
→ In questo caso viene fatto un rollback, si ripristina lo stato precedente l'attivazione del trigger.

Per l'**analisi della terminazione** si usa una rappresentazione delle regole detta **grafo di triggering**, che prevedere:

- Un nodo per ogni trigger;
- Un arco dal nodo t_i al nodo t_j , se l'esecuzione dell'azione di t_i può attivare il trigger t_j .

Se il grafo è **aciclico**, l'esecuzione termina sempre, se il grafo **ha cicli**, esso può avere problemi di terminazione.

Esempio con due trigger

T1:

```
create trigger AggiornaContributi
  after update of Stipendio on Impiegato
    update Impiegato
      set Contributi = Stipendio * 0.8
      where Matr in
        (select Matr from new_table)

  (mi permette di garantire un vincolo di
  tupla)
```

T2:

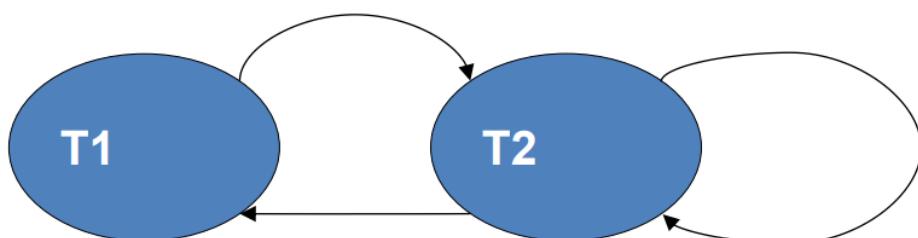
```
create trigger ControllaSogliaBudget
  after update on Impiegato
  when 50000 < (select
    (New.Stipendio + New.Contributi)
    from Impiegato)
    update Impiegato
      set Stipendio = 0.9 * Stipendio
```

Esempio con due trigger

Nel primo esempio ogni volta che si fa l'update dello Stipendio dell'impiegato, effettuo un'altra update e aggiorno i Contributi.

Nel secondo esempio ogni volta che si fa l'update in Impiegato, nel caso in cui lo Stipendio e i Contributi superano 50000, effettua un'altra update con cui si modifica lo Stipendio e lo si diminuisce.

Il grafo di triggering corrispondente è:

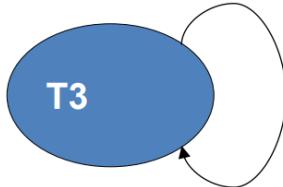


Ci sono due cicli ma il sistema termina sempre, sarebbe non terminante se in T2 il fattore di moltiplicazione fosse maggiore di 1 (se aumentassimo lo stipendio, perché questo continuerebbe a violare la condizione < 50000).

Esempio di non terminazione

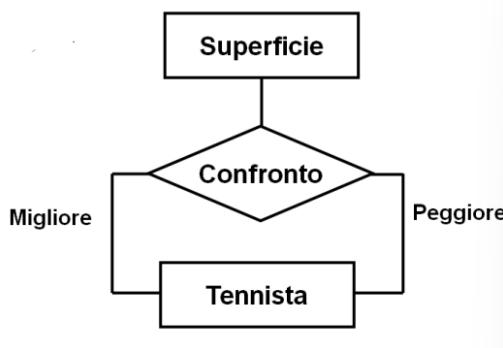
T3:

```
create trigger ControllaStipendio
    after update on Impiegato
    when New.Stipendio < 50000
    update Stipendio
    set Stipendio = 0.9 * Stipendio
```



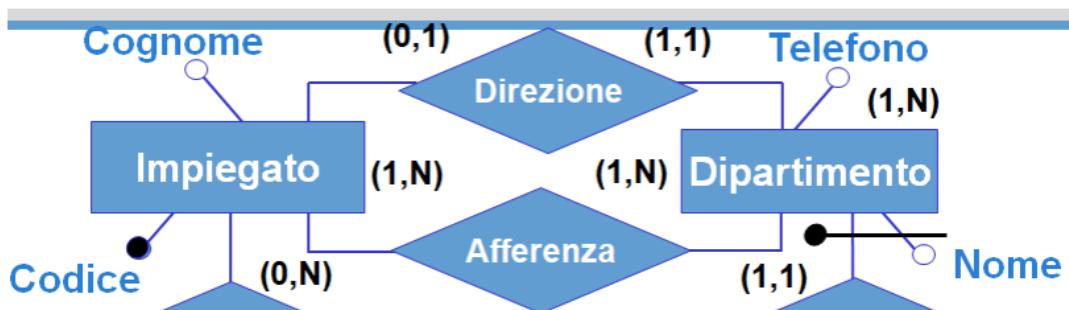
Un esempio di non terminazione si ha quando se lo stipendio è troppo basso viene diminuito ancora, questo porta a un ciclo.

Quali altri vincoli “non esprimibili in ER” potrei implementare tramite trigger?

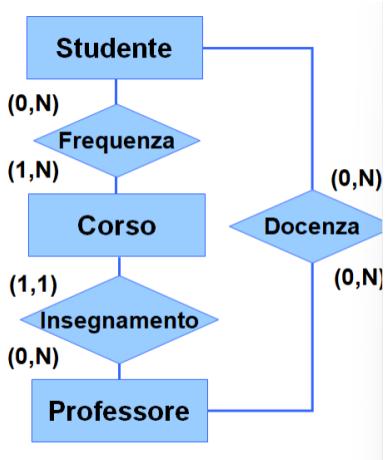


L’associazione Confronto è asimmetrica, dopo avere inserito la tupla “A migliore di B su S” controllo: se esiste un’altra tupla “B migliore di A su S”, allora cancello la nuova tupla.

Questo perché non ha senso che A sia il migliore e il peggiore nella stessa superficie.



Consideriamo il vincolo: “il direttore di un dipartimento deve afferire a quel dipartimento”. Dopo avere modificato o inserito la tupla “Il Dip. X ha come direttore l’impiegato Y”, controllo: se non ho una tupla (Y, X) nella relazione Afferenza, allora la aggiungo.



Dopo avere **modificato o inserito** la tupla Frequenza(Stud1,C1), seleziono il valore di Prof dalla tupla della tabella Corso di C1 (immaginiamo il prof sia Prof1), e verifico se esiste una tupla Docenza(Prof1, Stud1).

Se la tupla non esiste allora la aggiungo.

Nel DB abbiamo: Corso(C1, Prof1), viene aggiunta una nuova tupla in Frequenza(Stud1, C1). Il trigger che mi gestisce questo evento è:

```
create trigger AggiornaDocenza
```

```
after insert on Frequenza
```

```
insert into Docenza values (New.Studente,
                               (select Prof from Corso
                                where cod_corso = New.Corso))
```

Il trigger aggiunge la nuova tupla: Docenza(Prof1, Stud1).

A cosa servono i trigger? Posso fare tutto da applicazione! Ma i **trigger servono per implementare la logica dei dati, non delle applicazioni**. Permettono di implementare delle caratteristiche "invarianti" dei dati, che devono valere qualunque applicazione acceda ai dati.

CONTROLLO DELL'ACCESSO

In SQL è possibile specificare chi (utente) e come (lettura, scrittura, ...) può utilizzare la base di dati (o parte di essa).

Oggetto dei privilegi sono di solito le **tabelle**, ma anche altri tipi di risorse, quali **singoli attributi, viste o domini**. Un utente predefinito _system (amministratore della base di dati) ha tutti i privilegi, il creatore di una risorsa ha tutti i privilegi su di essa.

Un privilegio è caratterizzato da:

- La risorsa cui si riferisce;
- L'utente che concede il privilegio;
- L'utente che riceve il privilegio;
- L'azione che viene permessa;
- La trasmissibilità del privilegio.

I tipi di privilegi offerti da SQL sono:

- **insert**: permette di inserire nuovi oggetti (ennuple);
- **update**: permette di modificare il contenuto;
- **delete**: permette di eliminare oggetti;
- **select**: permette di leggere la risorsa;
- **references**: permette la definizione di vincoli di integrità referenziale verso la risorsa (può limitare la possibilità di modificare la risorsa da parte del proprietario, ad esempio con "no action");
- **alter**: permette modifiche allo schema;
- ecc..

Per concedere i privilegi si utilizza:

```
grant on Resource to Users [ with grant option ]
```

Dove **grant option** specifica se il privilegio può essere trasmesso ad altri utenti.

Per revocare i privilegi si utilizza:

```
revoke Privileges on Resource from Users [ restrict | cascade ]
```

Si usa **Restrict** se gli Users hanno concesso i privilegi ad altri, blocca la revoca, **Cascade** se gli Users hanno concesso i privilegi ad altri, propaga la revoca in cascata.

Esempi

- GRANT SELECT ON films TO PUBLIC;
- REVOKE INSERT ON films FROM PUBLIC;
(PUBLIC = tutti gli utenti)
- GRANT SELECT, UPDATE ON mytable TO bill;
- GRANT SELECT(title), UPDATE(title) ON films TO alice;
- GRANT ALL PRIVILEGES ON films TO bill;

La gestione delle autorizzazioni deve “nascondere” gli elementi cui un utente non può accedere, senza sospetti

Esempio:

- Impiegati non esiste;
- ImpiegatiSegreti esiste, ma l’utente non è autorizzato.

L’utente deve ricevere lo stesso messaggio se cerca di accedere alle due tabelle.

Come autorizzare un utente a **vedere solo alcune tuple di una relazione**? Attraverso una **vista**. Definiamo la vista con **una condizione di selezione**, attribuiamo le autorizzazioni sulla vista, anziché sulla relazione di base.

Ad ogni utente viene attribuito un **ruolo** con cui gli vengono concessi determinati privilegi.

ESEMPI:

- ↗ Attribuisci la membership al ruolo “amministratori” allo user “joe”:

```
GRANT SELECT on films TO amministratori;
GRANT amministratori TO joe;
```

- ↗ Creo un nuovo utente:

```
CREATE USER jim WITH PASSWORD 'jw8s0F4';
```

- ↗ Creo un nuovo ruolo:

```
CREATE ROLE amministratori;
```