

operazioni di undo e redo.

- **commit**: un record traccia che una transazione è stata completata con successo e il log tail viene scritto nello stable storage.
- **abort**: un record traccia una transazione abortita, e la transazione undo viene ripresa
- **end**: dopo un commit/abort, sono necessarie alcune operazioni di finalizzazione al fine della quale viene scritta una registrazione finale
- **undoing operation**: durante un recovery o durante l'undoing delle operazioni, viene un scritto un tipo speciale di file record, il Compensation Log Record (CLR). Un record CLR non viene mai ripristinato e traccia che un operazione è stata già annullata

## Checkpoint

Un checkpoint è uno snapshot dello stato del DB. I checkpoint riducono il tempo di ripristini. Invece di dover eseguire l'intero log file, è sufficiente eseguire all'indietro fino ad un checkpoint. ARIES crea checkpoints in tre step:

- **begin-checkpoint**: il record viene scritto nel log file
- **end-checkpoint**: il record, contenente *dirty page table* e *transaction table*, viene scritto nel log file
- alla fine dell'end-checkpoint il record viene scritto nello stable storage, un record, **Master Record**, contenente LSN del *begin-checkpoint* viene scritto in una parte conosciuta dello stable storage. Questo tipo di checkpoint viene detto **fuzzy checkpoint** e non è costoso in termini di performance. Non interrompe le normali operazioni del DBMS e non richiede la scrittura delle pagine del buffer pool.

## ARIES Crash Recovery

- **analysis phase**:
  - determina la locazione del log file da dove comincia la fase di redo, ovvero l'inizio dell'ultimo checkpoint
  - determina quale pagine del buffer pool contiene il dato modificato che non era ancora stato scritto al momento dell'anomalia.
  - identifica le transazioni che erano in corso al momento dell'anomalia
- **redo phase**: dalla *dirty page table*, ARIES identifica il minimo LSN di una dirty page. Da qui, ripete le azioni fino all'anomalia, nel caso in cui non erano già state rese persistenti
- **undo phase**: i cambiamenti di una transazione uncommitted devono essere annullate in modo da ripristinare il DB in uno stato consistente.

# Database Design

La progettazione è una delle attività che fanno parte del processo di sviluppo de sistemi informativi. Deve essere vista in un contesto più generale: il ciclo di vita dei sistemi informativi, comprende:

- **studio di fattibilità:** definisce i costi delle varie alternative possibili e le priorità di realizzazione delle componenti del sistema
- **raccolta e analisi dei requisiti:** individua e studia le proprietà e le funzionalità che il sistema informativo dovrà avere. E' necessaria l'interazione con l'utente per poter produrre una descrizione completa, generalmente informale, dei dati.
- **progettazione:**
  - *progettazione dei dati:* individua la struttura e l'organizzazione che i dati dovranno avere
  - *progettazione delle applicazioni:* definisce le caratteristiche dei programmi applicativi sono complementari e possono procedere in parallelo o in cascata. Le descrizioni dei dati e delle applicazioni diventano formali e fanno riferimento a modelli specifici.
- **implementazione:** realizzazione del sistema informativo. La base di dati viene costruita e popolata e viene prodotto il codice dei programmi;
- **funzionamento:** il sistema informativo diventa operativo.

Il processo non è quasi mai strettamente sequenziale, spesso durante l'esecuzione di una delle attività citate, bisogna rivedere decisioni prese nell'attività precedente.

Le basi di dati costituiscono solo una delle componenti di un sistema informativo, che comprende anche i programmi applicativi, le interfacce con l'utente e altri programmi di servizio. I dati però hanno un ruolo centrale.

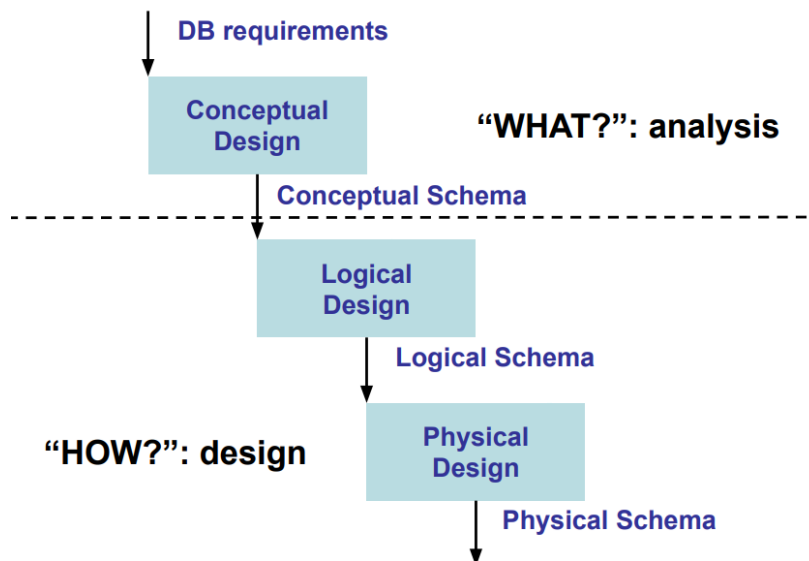
## Metodologie di progettazione

Una metodologia di progettazione consiste in:

- una *decomposizione* dell'attività di progetto in passi indipendenti tra loro
- delle *strategie* da seguire e alcuni criteri per la scelta in caso di alternative
- *modelli di riferimento* per descrivere i dati di ingresso e uscita delle varie fasi inoltre deve garantire:
- la *generalità* rispetto alle applicazioni e ai sistemi
- la *qualità del prodotto* in termini di correttezza, completezza ed efficienza
- la *facilità d'uso* delle strategie e dei modelli di riferimento

Tale metodologia è articolata in tre fasi principali: separare in maniera netta le decisioni relative a "cosa" rappresentare nella base di dati (prima fase), da quelle relative a "come" farlo (seconda e

terza fase).



- **progettazione concettuale:**

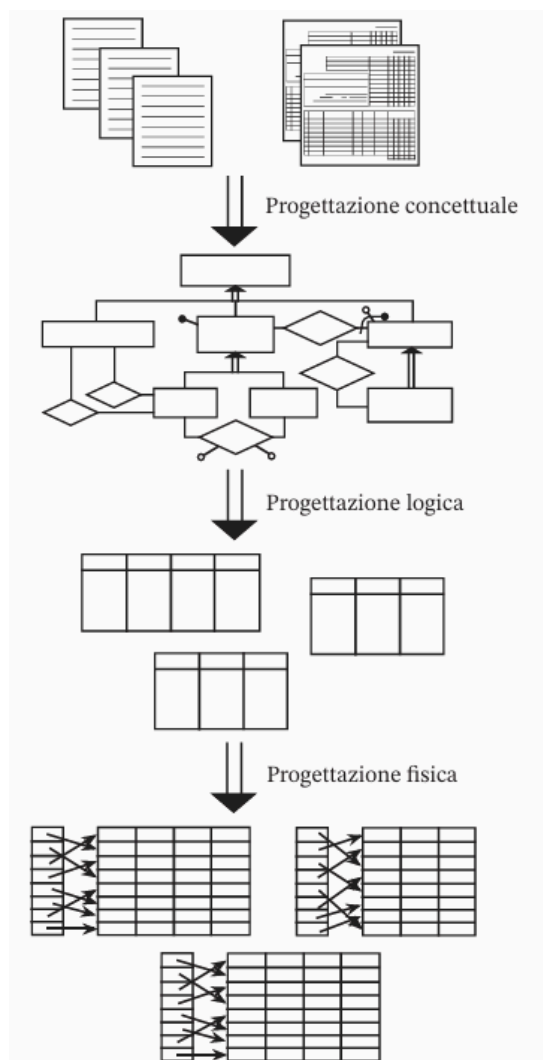
- rappresenta le specifiche informali della realtà di interesse con una descrizione formale e completa.
- Il prodotto di questa fase è lo *schema concettuale* che fa riferimento a un *modello concettuale*.
- Consente di descrivere l'organizzazione dei dati a un alto livello di astrazione, senza tenere conto degli aspetti implementativi, cercando di rappresentare il contenuto informativo della base di dati.

- **progettazione logica:**

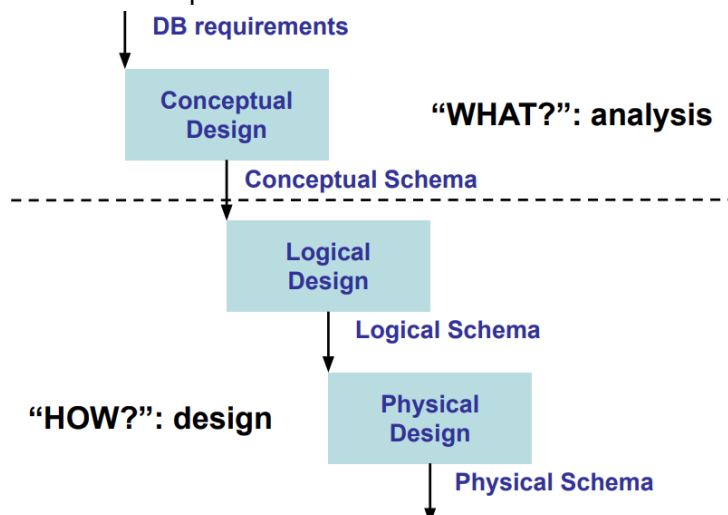
- traduzione dello schema concettuale in un modello di rappresentazione dei dati adottato dal sistema di gestione di base di dati a disposizione.
- Il prodotto di questa fase è lo *schema logico* che fa riferimento a un *modello logico*.
- Consente di scrivere i dati secondo una rappresentazione ancora indipendente da dettagli fisici, ma comunque concreta.
- Le scelte progettuali si basano su criteri di ottimizzazione delle operazioni da effettuare sui dati. (nel caso del modello relazionale dei dati, la tecnica comunemente utilizzata è quella della *normalizzazione*).

- **progettazione fisica:**

- lo schema logico viene completato con la specifica dei parametri fisici di memorizzazione dei dati (organizzazione dei file o degli indici)
- Il prodotto di questa fase è lo *schema fisico* e fa riferimento a un *modello fisico*



Distinguiamo le *specifiche sui dati*, che riguardano il contenuto della base di dati, da le *specifiche sulle operazioni*, ovvero l'uso che utenti e applicazioni fanno della base di dati. Nella progettazione concettuale si fa uso delle specifiche sui dati mentre le specifiche sulle operazioni servono a verificare che lo schema contenga le informazioni necessarie per eseguire tutte le operazioni previste. Nello schema logico, lo schema concettuale riassume le specifiche sui dati, mentre le specifiche sulle operazioni si utilizzano per ottenere uno schema logico efficiente. Bisogna conoscere il modello logico adottato ma non è necessario conoscere il DBMS che verrà usato. Nella progettazione fisica si fa uso dello schema logico e delle specifiche sulle operazioni per ottimizzare le prestazioni del sistema.

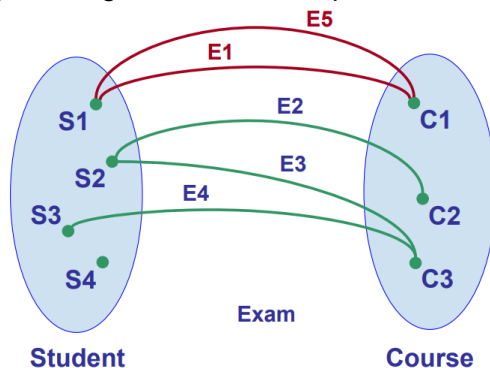


## Modello Entità-Relazione

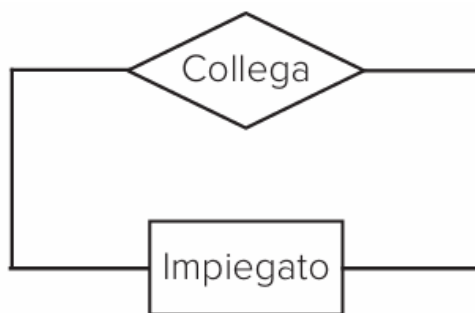
E' un modello concettuale di dati che fornisce dei *costrutti* per descrivere la realtà di interesse in maniera facile da comprendere e che prescinde dai criteri di organizzazione dei dati nei calcolatori.

### Costrutti Principali:

- **Entità:** rappresentano classi di oggetti che hanno proprietà comuni ed esistenza "autonoma" (cose, persone, posti). Ogni entità ha un nome unico e significativo.
- **Relazioni:**
  - rappresentano legami logici (associazioni) tra due o più entità.
  - L'insieme delle occorrenze di una relazione del modello E-R è una relazione matematica tra le occorrenze delle entità coinvolte, ovvero un sottoinsieme del loro prodotto cartesiano, questo significa che non ci possono essere ennuple ripetute.



- è possibile avere relazioni *ricorsive*, ovvero tra un'entità e se stessa.



- è possibile avere relazioni n-arie, ovvero che coinvolgono più di due entità
- talvolta può essere utile promuovere le relazioni a entità, ad esempio la relazione *esame* tra *Studente* e *Corso* non cattura la situazione in cui uno studente ha dato più volte lo stesso esame.
- **Attributi:**
  - descrivono le proprietà elementari di entità o relazioni che sono di interesse ai fini dell'applicazione;
  - un attributo associa a ciascuna occorrenza di entità (o di relazione) un valore appartenente a un insieme, detto *dominio*, che contiene i valori ammissibili per l'attributo (es. l'attributo cognome può avere come dominio l'insieme delle stringhe di 20 caratteri); i domini non vengono riportati nello schema, ma sono generalmente descritti nella documentazione associata;
  - è possibile raggruppare attributi di una medesima entità o relazione che presentano affinità nel loro significato ottenendo così un attributo composto

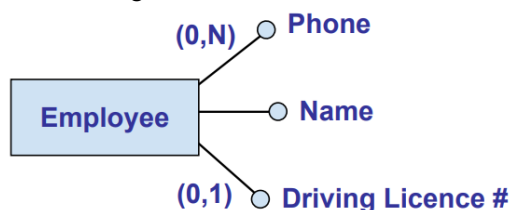


- **Cardinalità delle relazioni:**

- sono specificate per ciascuna partecipazione di entità a una relazione e descrivono il numero minimo e massimo di occorrenze di relazione a cui una occorrenza dell'entità può partecipare: quante volte in una relazione tra entità, un'occorrenza di una di queste entità può essere legata a occorrenze delle altre entità coinvolte;
- è possibile assegnare un qualunque intero non negativo a una cardinalità, con l'unico vincolo che la cardinalità minima deve essere minore o uguale della cardinalità massima:
  - per la cardinalità minima:
    - **0**: la partecipazione dell'entità relativa è opzionale;
    - **1**: la partecipazione è obbligatoria.
  - per la cardinalità massima:
    - **1**: la partecipazione dell'entità relativa può essere vista come una funzione che associa a una occorrenza dell'entità una sola occorrenza dell'altra entità;
    - **N**: c'è una associazione con un numero arbitrario di occorrenze dell'altra entità.
- *classificazione relazioni binarie in base alla cardinalità:*
  - **relazioni uno a uno**, relazioni aventi cardinalità massima pari a 1 per entrambe le entità coinvolte, corrispondenza uno a uno tra le occorrenze;
  - **relazioni uno a molti**: relazioni aventi un'entità con cardinalità massima pari a 1 e l'altra con cardinalità massima pari a N;
  - **relazioni molti a molti**: relazioni aventi cardinalità massima N per entrambe le entità coinvolte.

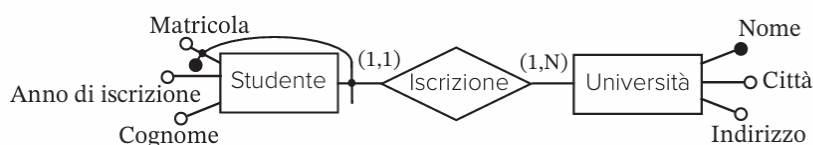
- **Cardinalità degli attributi:**

- possono essere specificate per gli attributi di entità o relazione e descrivono il numero minimo e massimo di valori dell'attributo associati a ogni occorrenza di entità o relazione.
- un attributo con cardinalità:
  - minima uguale a 0 è *opzionale*;
  - minima uguale a 1 è *obbligatorio*;
  - massima uguale a N è *multivalore*.



- **Identificatori delle entità**

- vengono specificati per ciascuna entità e descrivono i concetti (attributi e/o entità) che permettono di identificare in modo univoco le occorrenze delle entità.
- due casi:
  - **identificatore interno**: uno o più attributi di un'entità sono sufficienti a individuare un identificatore;
  - **identificatore esterno**: gli attributi dell'entità non sono sufficienti e l'identificatore è ottenuto utilizzando altre entità.



- considerazioni generali:
  - un identificatore può coinvolgere uno o più attributi, ognuno dei quali deve avere cardinalità (1,1);
  - un'identificazione esterna può coinvolgere una o più entità, ognuna delle quali deve essere membro di una relazione alla quale l'entità da identificare partecipa con cardinalità (1,1);
  - un'identificazione esterna può coinvolgere un'entità che è a sua volta identificata esternamente, purché non vengano generati cicli di identificazioni esterne;
  - ogni entità deve avere almeno un identificatore (interno o esterno), ma ne può avere in generale più di uno; nel caso di più identificatori, le restrizioni indicate possono essere rilassate in quanto gli attributi e le entità coinvolte in alcune identificazioni, tranne una, possono essere opzionali (cardinalità minima uguale a 0).
- **Generalizzazioni:**
  - rappresentano legami logici tra un'entità E, detta *genitore*, e una o più entità  $E_1, \dots, E_n$ , dette entità *figlie*, di cui E è più generale, nel senso che le comprende come caso particolare.
  - E è *generalizzazione* di  $E_1, \dots, E_n$  e le entità  $E_1, \dots, E_n$  sono *specializzazioni* dell'entità E.
  - Tra le entità coinvolte in una generalizzazione valgono le seguenti proprietà:
    - ogni occorrenza di un'entità figlia è anche un'occorrenza dell'entità genitore;
    - ogni proprietà dell'entità genitore (attributi, identificatori, relazioni e altre generalizzazioni) è anche proprietà delle entità figlie → *ereditarietà*;
  - possono essere classificate sulla base di due proprietà tra loro ortogonali:
    - **generalizzazione totale**, se ogni occorrenza dell'entità genitore è un'occorrenza di almeno una delle entità figlie, altrimenti è **parziale**;
    - **generalizzazione esclusiva**, ogni occorrenza dell'entità genitore è al più un'occorrenza di una delle entità figlie, altrimenti è **sovrapposta**.

es.

*Persona* ⇒ *Uomo* e *Donna*: totale ed esclusiva;

*Professionista* ⇒ *Ingegnere* e *Dottore*: parziale ed esclusiva, si presume che ciascun professionista abbia una sola professione principale e che ci siano altre professioni;

*Persone* ⇒ *Studente* e *Lavoratore*: parziale e sovrapposta, esistono studenti che sono anche lavoratori.

  - una stessa entità può essere coinvolta in più generalizzazioni diverse.
  - possono esserci generalizzazioni su più livelli: *gerarchia* di generalizzazioni.
  - una generalizzazione può avere una sola entità figlia: *sottoinsieme*.

## Documentazione di schemi E-R

Uno schema E-R non è quasi mai sufficiente da solo, in quanto:

- compaiono solo i nomi dei vari concetti in esso presenti e può non essere sufficiente per comprenderne il significato;
  - nel caso di schemi particolarmente complessi potremmo non riuscire a rappresentare in maniera comprensibile ed esaustiva i concetti;
  - non si possono rappresentare *vincoli di integrità sui dati*.
- Diventa indispensabile corredare ogni schema E-R con una documentazione di supporto.

## Regole Aziendali

Le regole aziendali, o *business rules*, sono uno strumento utilizzato per la descrizione delle proprietà di un'applicazione che non si riescono a rappresentare con modelli concettuali. Una regola aziendale può essere:

- *descrizione di un concetto* rilevante per l'applicazione, ovvero la definizione precisa di un'entità, di un'attributo o di una relazione dello schema;
- *vincolo di integrità* sui dati dell'applicazione;
- una *derivazione*, un concetto che può essere ottenuto, attraverso un'inferenza o un calcolo aritmetico, da altri concetti dello schema.

Le regole che descrivono vincoli di integrità possono essere espresse sotto forma di asserzioni, affermazioni che devono essere sempre verificate nella base di dati:

< *concetto* > *deve/non deve* < *espressione su concetti* >

es. (RV) un impiegato *non deve* avere uno stipendio maggiore del direttore del dipartimento al quale afferisce.

Le regole aziendali che esprimono derivazioni possono essere espresse specificando le operazioni:

< *concetto* > *si ottiene* < *operazione su concetti* >

es (RD) il numero degli impiegati di un dipartimento *si ottiene* contando gli impiegati che vi afferiscono.

## Tecniche di documentazione

Le regole aziendali di tipo descrittivo possono essere rappresentato facendo uso di un *dizionario dei dati* in cui troviamo due tabelle:

1. descrive le entità dello schema con il nome, una definizione informale in linguaggio naturale, l'elenco di tutti gli attributi e i possibili identificatori;

Entity	Description	Attributes	Identifier
Employee	Employee in a Dept.	Code, Name, Surname	Code
Project	Projects of a Dept.	Name, Budget	Name
Department	Structure of a Dept.	Name, Phone	Name, Office
Office	Office's location	City, Address	City

2. descrive le relazioni con il nome, una loro descrizione informale, l'elenco degli attributi e l'elenco delle entità coinvolte insieme alla loro cardinalità.

Relationships	Description	Components	Attributes
Management	Management of a Dept.	Employee, Dept.	
Affiliation	Affiliation to a Dept.	Employee, Dept.	Date
Attendance	Attendance for a project	Employee, Project	
Composition	Composition of Departments	Dept., Office	

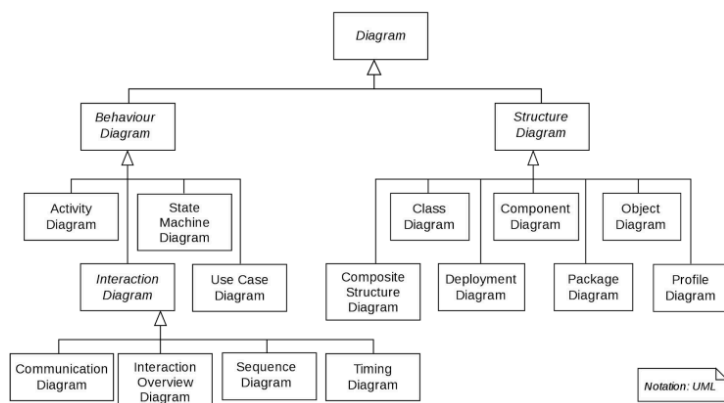


Possiamo anche avere una tabella per rappresentare i vincoli:

Integrity Constraints on Data	
1.	A department director must belong to that department
2.	An employee must not have an income greater than the director of the department which he is affiliated
3.	A department placed in Rome must be directed by an employee with at least 10 years of service
4.	An employee that isn't affiliated to any department must not attend to any project

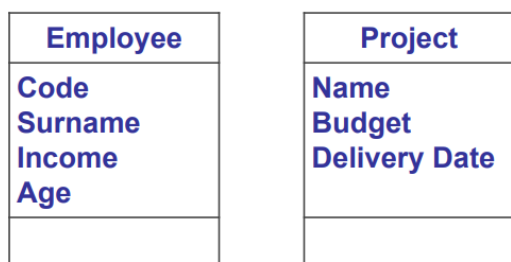
## Modellazione dei dati in UML

UML (Unified Modeling Language) è un linguaggio grafico per la modellazione di applicazioni software basate sulla programmazione orientata agli oggetti. Utilizza i *diagrammi delle classi* che descrivono le classi di oggetti di interesse per l'applicazione e le relazioni che intercorrono tra di esse. Inoltre in base al principio di *incapsulamento* della oop, prevede una stretta correlazione tra dati e operazioni; infatti è possibile rappresentare oltre agli aspetti strutturali dell'applicazione, ovvero i dati sui quali opera, anche quelli "comportamentali", ovvero le procedure associate ai dati.



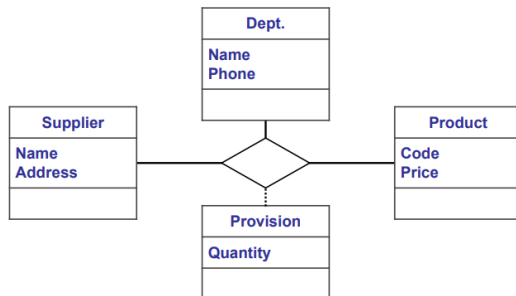
## Rappresentazione dei dati con i diagrammi delle classi

- **Classi:**
  - componenti principali dei diagrammi delle classi;
  - corrispondono alle entità del modello E-R;
  - rappresentato come un rettangolo contenente:
    - in alto: il nome della classe;
    - all'interno: gli attributi associati alla classe;
    - in basso è possibile specificare i relativi *metodi*, ovvero le operazioni ammissibili su oggetti della classe secondo il principio di incapsulamento.

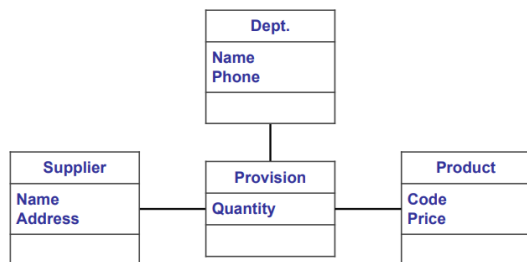


- **Associazioni:**
  - corrispondono alle relazioni del modello E-R;

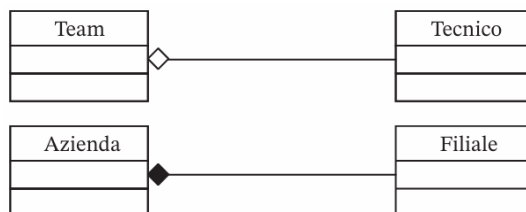
- le associazioni binarie si rappresentano con semplici linee che congiungono le classi coinvolte, il nome è posto sulla linea, ma non è obbligatorio perché in UML possono esistere associazioni senza nome;
- è possibile associare ruoli alle classi coinvolte in un'associazione;
- non è possibile assegnare attributi alle associazioni, per farlo si fa uso delle *classi di associazioni*;
- se l'associazione è n-aria, viene rappresentata da un rombo e da linee che congiungono il rombo con le classi che partecipano all'associazione;



le associazioni n-arie vengono usate di rado e viene suggerito di reificarle, ovvero trasformarle in una classe legata alle classi originarie con associazioni binarie;



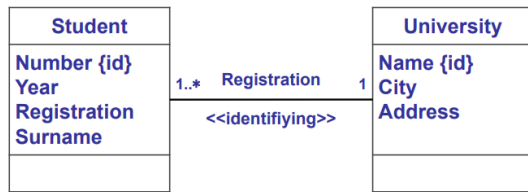
- **aggregazioni** di concetti:
  - sono associazioni che definiscono una relazione tra un concetto composto e uno o più concetti che ne costituiscono una sua parte;
  - si indicano con una linea avente un rombo attaccato alla classe che rappresenta il concetto "aggregante"; dall'altro capo della linea c'è una classe che costituisce una sua "parte";
  - il rombo si lascia in bianco se un oggetto della classe "parte" può esistere senza dover appartenere a un oggetto della classe "aggregante", altrimenti viene annerito e l'aggregazione viene chiamata *composizione*.



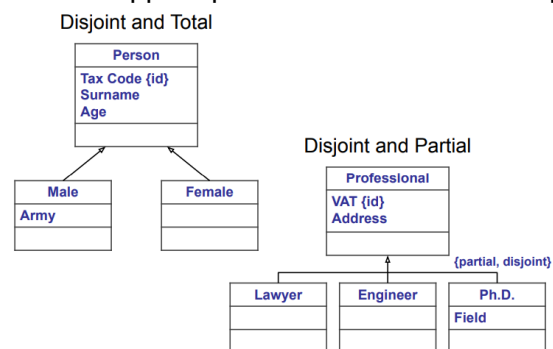
- **Molteplicità:**
  - è possibile indicare le cardinalità di partecipazione come coppia di valori che specificano la cardinalità minima e massima di partecipazione di un oggetto della classe dell'associazione;
  - la cardinalità minima viene separata dalla massima da due punti (0..1);
  - quando si specifica solo *si intende 0..*, ovvero (0, N);
  - quando si specifica solo 1 si denota la coppia di cardinalità 1..1, considerata quella di default per le classi.
- **Identificatori:** non esiste una notazione per esprimere identificatori di classi, secondo il paradigma oop ogni oggetto è dotato implicitamente di un identificatore che ne consente l'identificazione

univoca. D'altronde, gli identificatori sono indispensabili nel modello relazionale e perciò introduciamo il *vincolo utente*, ovvero la possibilità di definire vincoli di integrità su associazioni e su attributi specificandoli tra parentesi graffe vicino all'elemento oggetto del vincolo.

- gli attributi che compongono un identificatore sono indicati con il vincolo utente {id};
- è possibile specificare un solo identificatore per classe;
- le identificazioni esterne, dato che sintatticamente non è possibile usare un vincolo, si ricorre a uno stereotipo: indicato da un nome racchiuso tra i simboli <<>>.



- **Generalizzazioni:** esiste in UML la possibilità di definire generalizzazioni, eventuali proprietà possono essere rappresentate con vincoli; in particolare possiamo indicare se la generalizzazione è totale oppure parziale e se è esclusiva oppure sovrapposta.



# Conceptual Design

L'analisi dei requisiti non è standardizzabile ma ci sono alcune regole pratiche nella fase di sviluppo della base di dati.

- *raccolta dei requisiti*: completa individuazione dei problemi che l'applicazione deve risolvere e le caratteristiche che dovrà avere;
- *analisi dei requisiti*: organizzazione delle specifiche dei requisiti.  
Principali fonti di informazione:
  - *gli utenti dell'applicazione*, mediante le interviste oppure attraverso una documentazione scritta;
  - *documentazione esistente*, moduli, regolamenti interni, procedure aziendali e normative. E' richiesta un'attività di raccolta e selezione che viene assistita dagli utenti a carico del progettista;
  - *realizzazioni preesistenti*, applicazioni che si devono rimpiazzare o che devono interagire con il sistema da realizzare.

Alcune regole generali:

- *scegliere il corretto livello di astrazione*: evitare termini troppo generici o troppo specifici;
- *standardizzare la struttura delle frasi*: è preferibile utilizzare sempre lo stesso stile sintattico;
- *evitare frasi contorte*: le definizioni devono essere semplici e chiare;
- *individuare sinonimi/omonimi e unificare i termini*: l'uso di sinonimi e omonimi può generare ambiguità e perciò si tende ad utilizzare un termine unico per i sinonimi, mentre nel caso di omonimi si utilizzano termini diversi;
- *rendere esplicito il riferimento tra termini*: l'assenza di un contesto di riferimento può rendere alcuni concetti ambigui, allora occorre esplicitare il riferimento tra termini;
- *costruire un glossario dei termini*: utile per la comprensione e la precisazione dei termini usati, deve contenere per ogni termine una breve descrizione, possibili sinonimi e altri termini nel glossario con i quali esiste un legame logico.

## Design Pattern

Ci sono alcune buone pratiche per una corretta rappresentazione concettuale dei dati, come dei *criteri generali di rappresentazione* o *design pattern*, ovvero soluzioni progettuali a problemi comuni.

### Criteri generali di rappresentazione

Premessa, non esiste una rappresentazione univoca di un insieme di specifiche, ma è utile avere delle indicazioni sulle scelte più opportune:

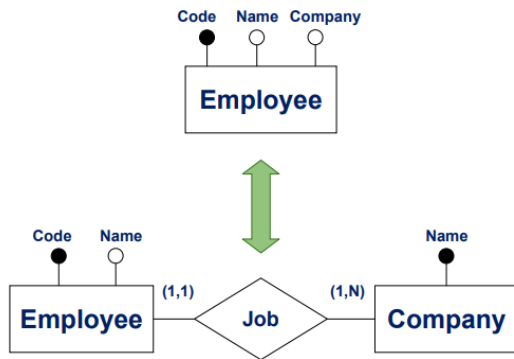
- se un concetto ha proprietà significative e/o descrive classi di oggetti con esistenza autonoma, è opportuno rappresentarlo con un'entità;
- se un concetto ha una struttura semplice e non possiede proprietà rilevanti associate è opportuno rappresentarlo con un attributo di un altro concetto a cui si riferisce;
- se sono state individuate due (o più) entità e nei requisiti compare un concetto che le associa, questo concetto può essere rappresentato da una relazione;

- se uno o più concetti risultano essere casi particolari di un altro, è opportuno rappresentarli facendo uso di una generalizzazione.
- Questi criteri hanno validità generale, cioè sono indipendenti dalla strategia di progettazione scelta.

## Pattern di Progetto

### 1. Reificazione degli attributi:

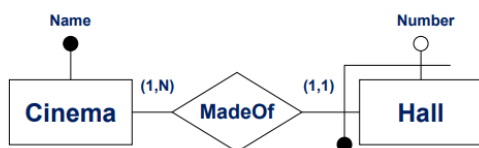
Viene rappresentato un *impiegato* con codice, nome e azienda, ma non stiamo rappresentando anche il concetto di *azienda*; per poterlo rappresentare esplicitamente reifichiamo l'attributo facendolo diventare un'entità.



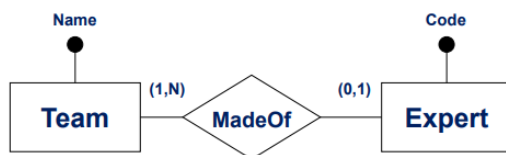
### 2. Part-Of

Si vuole rappresentare il fatto che un'entità è parte di un'altra entità, tipicamente queste relazioni sono uno a molti e si rappresentano in due forme:

- l'esistenza di un'occorrenza dell'entità "parte" dipende dall'esistenza di un'occorrenza dell'entità che la contiene:



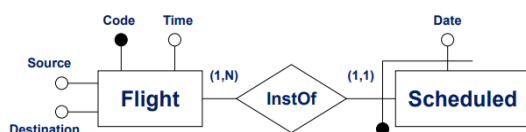
- l'entità contenuta nell'altra ha esistenza autonoma:



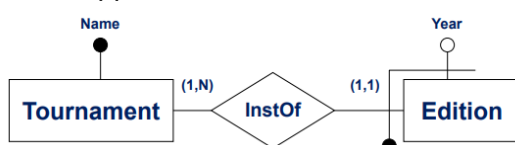
### 3. Instance-Of:

Talvolta sono necessarie due entità distinte, una che coinvolgono la rappresentazione astratta e un'altra che memorizza le informazioni necessarie per i nostri requisiti.

- un'entità che descrive il concetto astratto di volo e un'altra entità che rappresenta il volo reale:

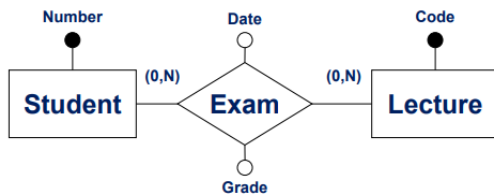


- viene rappresentato il concetto di torneo sportivo e una sua edizione:

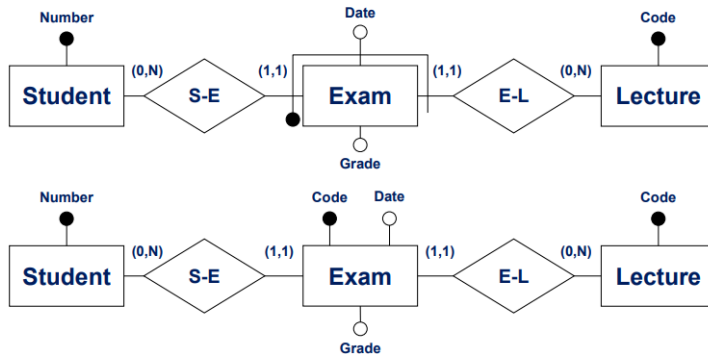


### 4. Reification: Binary Relations:

si utilizza una relazione, tipicamente molti a molti, per descrivere un concetto che lega altri due concetti, ad esempio: rappresentiamo un'esame come relazione tra *studente* e *corso*:

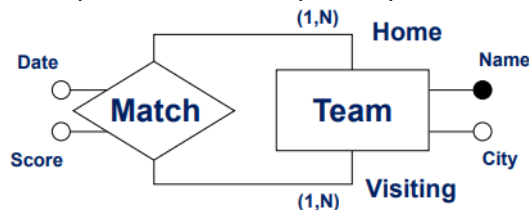


Questa soluzione è valida solo se ogni studente può sostenere una sola volta un certo esame (per def: una occorrenza della relazione *esame* è un insieme di coppie *studente-corso*, senza duplicati). La soluzione corretta si ottiene reificando la relazione *esame* rappresentandola come entità:

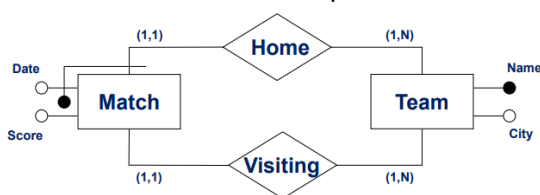


## 5. Reification: Recursive Relation:

Esempio: il concetto di *partita* può essere visto come una relazione ricorsiva sull'entità squadre,



ma se in un torneo due squadre si incontrano più volte è necessario reificare la relazione binaria:

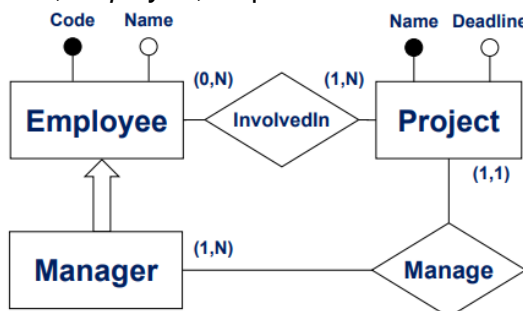


L'identificazione dell'entità *partita* coinvolge solo la data e la squadra che gioca in casa perché si assume che una squadra non possa giocare due partite nello stesso giorno.

## 6. Generalizzazioni:

- *specific case*:

Le generalizzazioni sono usate per definire casi specifici, nell'esempio *manager*, dell'entità *data*, *employee*; in questo caso non tutte gli impiegati gestiscono un progetto:

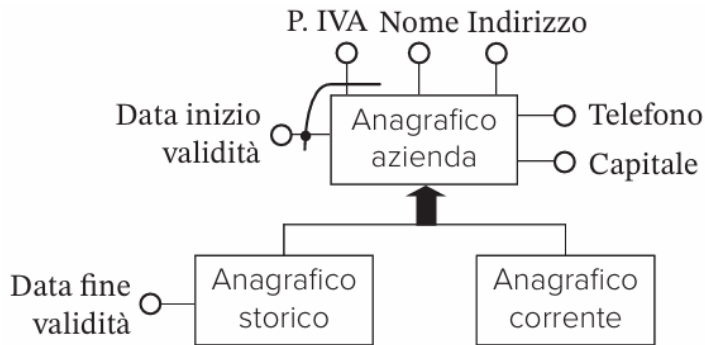


- *storicizzazione di un'entità*:

esempio, vogliamo memorizzare le informazioni correnti di un'azienda, tenendo traccia dei dati che sono variati.

soluzione: utilizzare due entità con gli stessi attributi, una rappresenta il concetto di interesse con le informazioni aggiornate e l'altra lo storico. Le proprietà di queste entità vengono messe

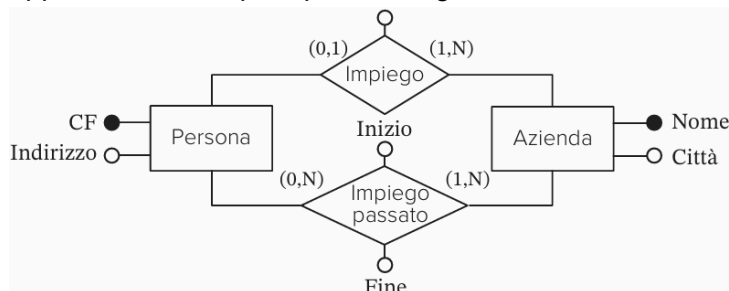
a fattor comune mediante una generalizzazione la cui entità genitore rappresenta tutte le informazioni anagrafiche delle aziende, sia quelle correnti sia quelle passate; in più vengono introdotti degli attributi per definire l'intervallo di validità dei dati (data inizio e data fine). L'identificazione si ottiene aggiungendo alla chiave originaria (p. iva) la data di inizio, istante che diventerà anche la data di fine validità delle informazioni che vengono soppiantate.



- **storicizzazione di una relazione:**

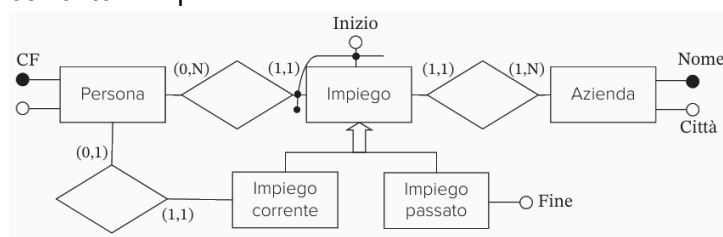
esempio, vogliamo rappresentare gli impieghi presenti e passati di una persona.

**soluzione uno:** rappresentare separatamente i dati correnti e i dati storici e introdurre opportuni attributi per specificare gli intervalli di validità.



Ma, osservando le cardinalità delle partecipazioni dell'entità *persona* alle due relazioni, vediamo che non possiamo rappresentare il fatto che una persona possa aver lavorato, in periodi diversi, per la stessa azienda.

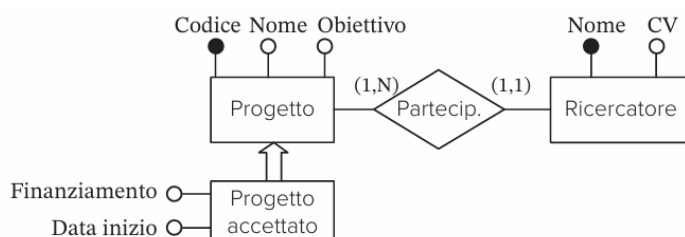
**soluzione due:** reificazione delle relazioni, inoltre è necessario l'inserimento di un vincolo esterno allo schema che impone che tutte le occorrenze della relazione tra *persona* e *impiego corrente* compaiano anche tra le occorrenze della relazione tra *persona* e *impiego*.



- **estendere un concetto:**

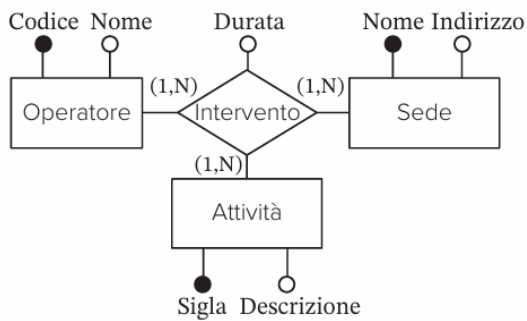
vogliamo rappresentare il fatto che un certo concetto subisce un'evoluzione nel tempo che può essere diversa per le diverse occorrenze del concetto.

esempio: si vuole rappresentare progetti che vengono proposti con l'obiettivo di ottenere un finanziamento. Solo alcuni vengono accettati e per questi vengono aggiunte ulteriori informazioni: data di inizio ufficiale e finanziamento.



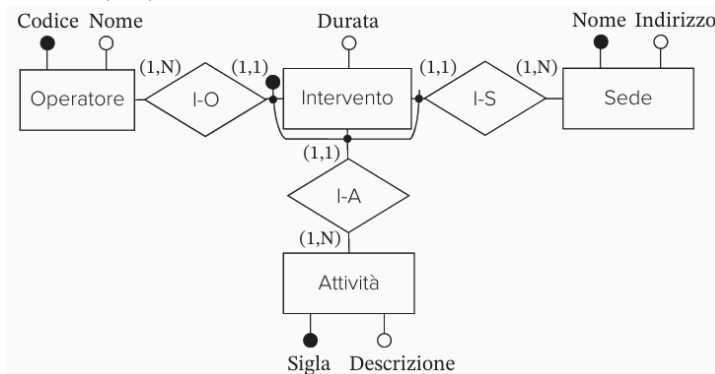
- **Relazioni ternarie:**

*esempio 1:* si vuole modellare il caso in cui un operatore può effettuare operazioni che consistono in attività diverse svolte in sedi diverse, in ogni sede possono operare operatori diversi svolgendo attività diverse e le attività possono essere svolte da operatori diversi e in sedi diverse.

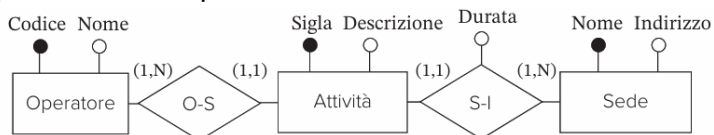


Questo schema può essere reificato per modellare altre realtà:

- Ogni lavoro è definito da un *Operatore* (I-O) che lavora in una certa *Sede* (I-S) per una certa *Attività* (I-A).



- Se un'attività può essere eseguita da un solo operatore in un solo ufficio, allora la reificazione può essere semplificata.

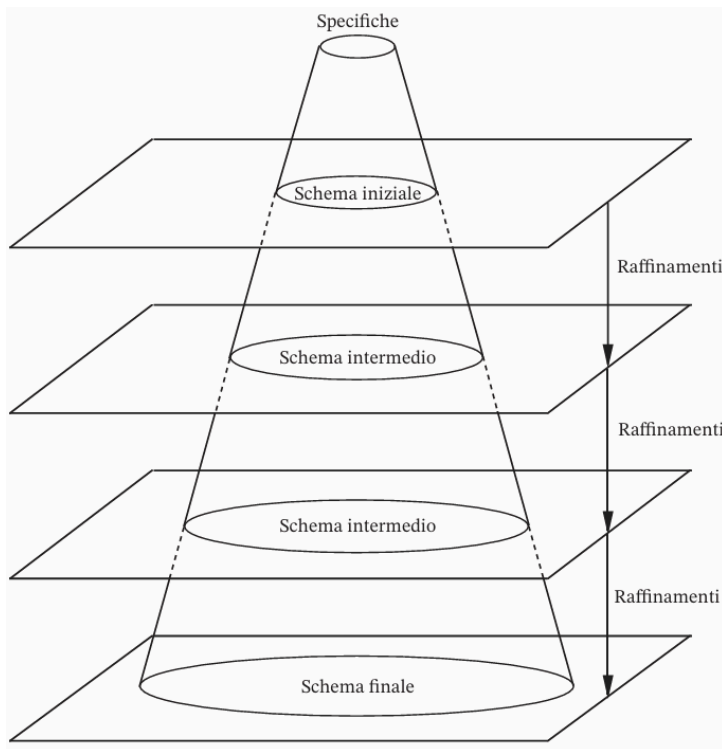


## Strategie di Progetto

### Strategia top-down

Si parte da uno schema iniziale che descrive tutte le specifiche con pochi concetti molto astratti, questo schema viene poi via via raffinato mediante trasformazioni che aumentano il dettaglio dei vari concetti.





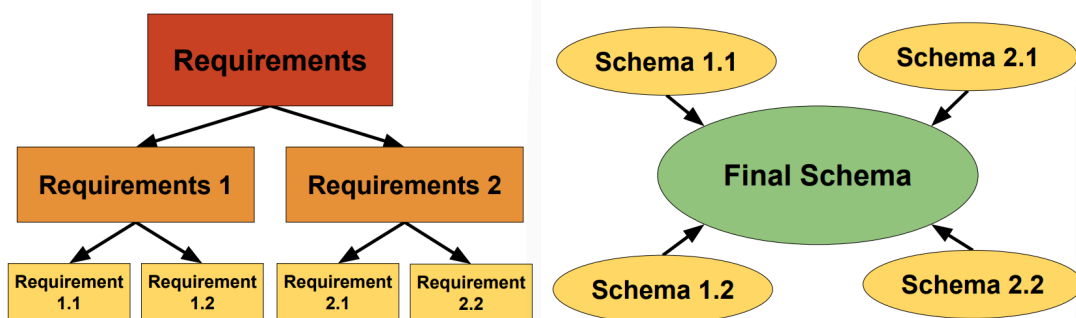
Ognuno di questi piani descrive le stesse informazioni a un diverso livello di dettaglio. Con questa strategia tutti gli aspetti presenti nello schema finale sono presenti, in linea di principio a ogni livello di raffinamento. Il vantaggio sta nel poter descrivere inizialmente tutte le specifiche dei dati trascurandone i dettagli, per poi entrare nel merito di un concetto alla volta. Questo però è possibile solo quando si possiede una visione globale di tutte le componenti del sistema.

## Strategia bottom-up

Le specifiche iniziali sono suddivise in componenti via via sempre più piccole. Le varie componenti vengono rappresentate da semplici schemi concettuali, questi vengono poi fusi fino a giungere allo schema concettuale finale. Questo schema si ottiene attraverso alcune trasformazioni elementari, denominate *primitive di trasformazione bottom-up*.

**Vantaggio:** si adatta a una decomposizione del problema in componenti più semplici, affrontabili in parallelo da progettisti diversi.

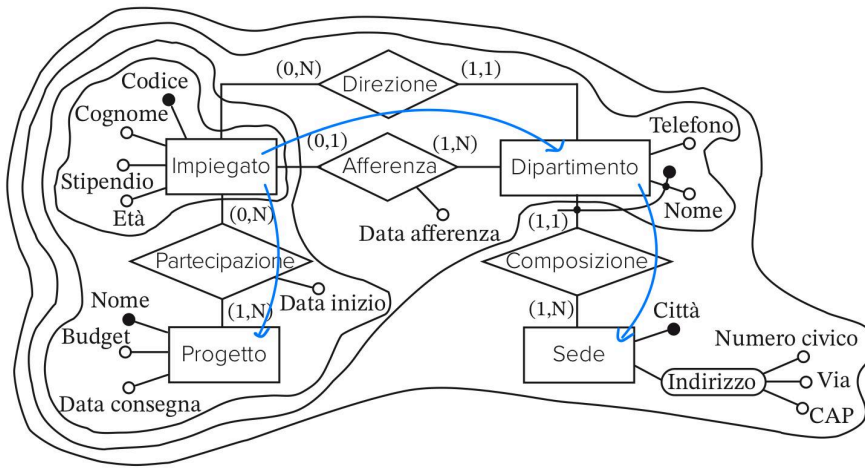
**Svantaggio:** richiede operazioni di integrazione di schemi concettuali diversi, che in caso di problemi complessi, presentano grosse difficoltà.



## Strategia inside-out

Può essere vista come un caso particolare della strategia bottom-up. Si individuano inizialmente solo alcuni concetti importanti e poi si procede, a partire da questi. Si rappresentano prima i concetti in

relazione con i concetti iniziali, per poi muoversi verso quelli più lontani.



## Rule of Thumb

Usare sempre un approccio misto:

- prima, si crea una bozza utilizzando le entit  pi  rilevanti;
- successivamente, si decompone lo schema;
- infine, raffinare (top-down), integrare (bottom-up) ed espandere (inside-out).

## Metodologia Generale

### 1. Analisi dei requisiti:

- costruire un glossario dei termini;
- analizzare i requisiti ed eliminare le ambiguit  presenti;
- raggruppare i requisiti in insiemi omogenei.

### 2. Passo Base:

- individuare i concetti pi  rilevanti e rappresentarli in uno schema scheletro.

### 3. Passo di decomposizione (da effettuare se necessario):

- effettuare una decomposizione dei requisiti con riferimento ai concetti presenti nello schema scheletro.

### 4. Passo iterativo, da ripetere per tutti i sottoschemi (se presenti) finch  ogni specifica   stata rappresentata:

- raffinare i concetti presenti sulla base delle loro specifiche;
- aggiungere nuovi concetti allo schema per descrivere specifiche non ancora descritte.

### 5. Passo di integrazione (da effettuare solo se   stato eseguito il passo 3):

- integrare i vari sottoschemi in uno schema generale facendo riferimento allo schema scheletro.

### 6. Analisi di qualit :

controllare la qualit  dello schema e modificarlo.

## Qualit  di uno schema concettuale

- **Correttezza** → quando utilizza propriamente i costrutti messi a disposizione.
- **Completezza** → quando rappresenta tutti i dati di interesse e quando tutte le operazioni possono essere eseguite a partire dai concetti descritti nello schema.

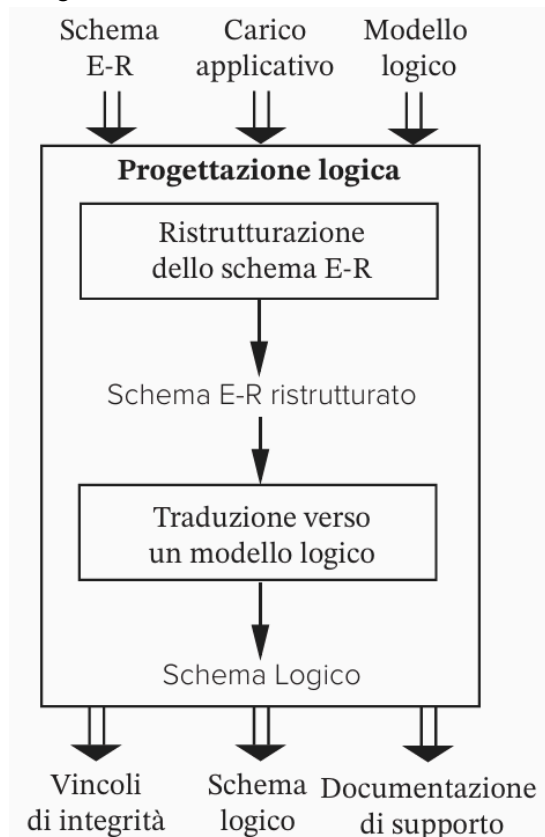
- **Leggibilità** → quando rappresenta i requisiti in maniera naturale e facilmente comprensibile (es. mediante una scelta opportuna dei nomi da dare ai concetti).
- **Minimalità** → quando tutte le specifiche sui dati sono rappresentate una volta sola nello schema (es fonte di ridondanza: presenza dei cicli dovuta alla presenza di relazioni e/o generalizzazioni).

# Logical Design

Le attività principali della progettazione logica sono:

- *ristrutturazione dello schema Entità-Relazione*: si basa su criteri di ottimizzazione dello schema e di semplificazione della fase successiva;
- *traduzione verso il modello logico*: fa riferimento a uno specifico modello logico e può includere un'ulteriore ottimizzazione che si basa sulle caratteristiche del modello logico stesso.

I dati di ingresso sono lo schema concettuale e il carico applicativo previsto, ovvero la dimensione dei dati e le caratteristiche delle operazioni. Il risultato è uno schema E-R ristrutturato. Questo schema e il modello logico scelto sono i dati di ingresso della seconda fase che produce lo schema logico; in questa fase si fanno verifiche della qualità dello schema ed eventuali ulteriori ottimizzazioni. I prodotti finali della progettazione logica sono lo schema logico finale, i vincoli di integrità definiti su di esso e la relativa documentazione.



## Analisi delle prestazioni

Uno schema E-R può essere modificato per ottimizzare alcuni *indici di prestazione*, chiamati così perché non sono valutabili in maniera precisa in sede di progettazione logica in quanto sono dipendenti anche da parametri fisici, ecc..

- *costo di un'operazione*: valutato in termini di numero di occorrenze di entità e associazioni che mediamente vanno visitate per rispondere a un'operazione sulla base di dati;
- *occupazione di memoria*: valutato in termini dello spazio di memoria necessario per memorizzare i dati descritti dallo schema.

Per lo studio di questi parametri serve conoscere:

- *volume dei dati*: