La progettazione delle basi di dati

Progettazione Concettuale

Prof. Fedeli Massimo

IIS Fermi Sacconi Ceci - Ascoli Piceno

November 29, 2025

La progettazione del database

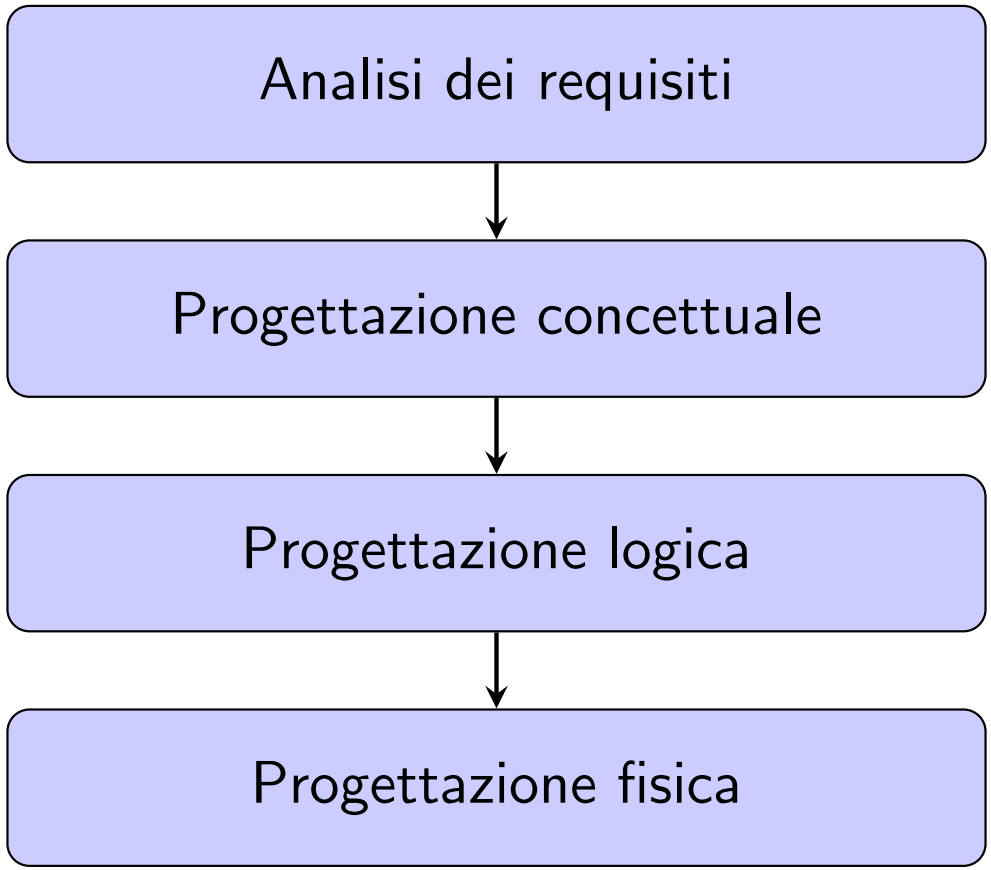
I passi principali per progettare un database si possono così schematizzare:

1. Analisi del problema
2. Progettazione concettuale del database $\rightarrow$ modello E-R
3. Progettazione logica del database $\rightarrow$ schema logico
4. Progettazione fisica e implementazione
5. Realizzazione delle applicazioni $\rightarrow$ (Java, VB, PHP, etc)

Ognuno di questi passi presenta delle criticità e implica un insieme di operazioni anche complesse tra cui:

- la fase di sviluppo del database
- la fase di sviluppo dell'applicazione
- i test di funzionamento di entrambi, il collaudo ecc

Le fasi della progettazione di un database

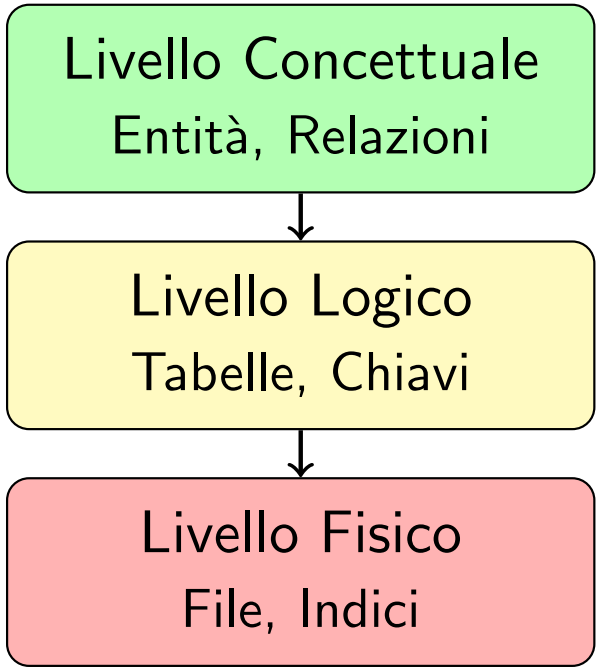


L'output della progettazione concettuale è il **modello concettuale**.

I livelli di astrazione del modello di database

La progettazione di un modello di dati avviene a diversi livelli di astrazione:

- **Livello concettuale**
- **Livello logico**
- **Livello fisico**



Il livello fisico è l'implementazione del livello logico sui supporti per la registrazione fisica dei dati (partizioni, puntatori, blocchi fisici, cluster, indici).

Analisi preliminare alla modellazione

L'analisi preliminare per la modellazione dei dati avviene solitamente cercando di individuare:

- le esigenze del cliente
- il dominio dell'applicazione
- quali informazioni devono essere salvate
- in che modo queste informazioni verranno manipolate dall'utente

Le tecniche di analisi del problema sono da considerarsi valide e applicabili anche alla progettazione dei database: naturalmente bisognerà prestare maggiore attenzione al dato e quindi spesso la tecnica **bottom-up** è da preferirsi a quella top-down.

La fase di modellazione

Al termine dell'analisi inizia la prima fase di modellazione, che è quella concettuale; per attuarla, si può far ricorso ai due seguenti modelli:

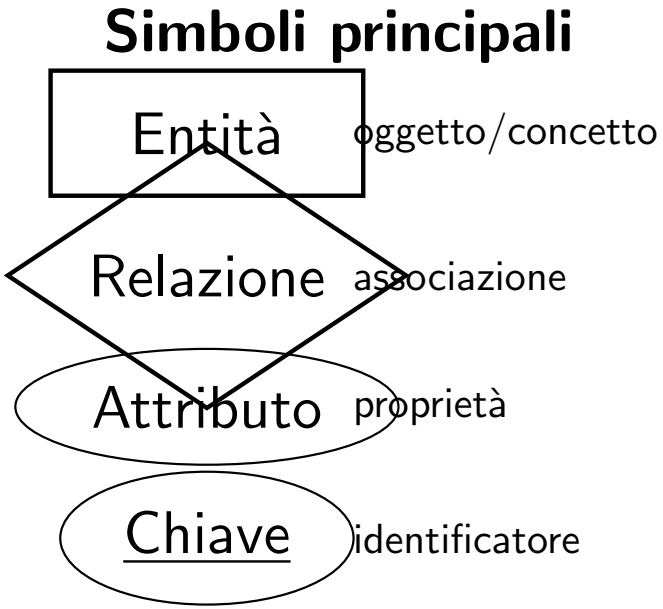
- modello entità-relazione
- modello a oggetti

Benché il secondo modello stia cominciando a rivelarsi interessante da un punto di vista commerciale e non solamente accademico, la stragrande maggioranza delle applicazioni esistenti ricorre all'approccio **Entità-Relazione (E-R)**.

Modello entità-relazione

Il modello Entità-Relazione consente di superare questo ostacolo in quanto è assolutamente indipendente dal linguaggio scritto o parlato e permette quindi a tutti di comprendere la struttura del database.

Di norma, al modello Entità-Relazione viene affiancato un **documento tecnico** che descrive in maggior dettaglio i concetti espressi graficamente.



Progettazione concettuale → modello entità-relazione

- Il primo scopo del modello Entità-Relazione è quello di fornire la **rappresentazione grafica** di tutti gli oggetti che fanno parte di un database così che il flusso delle informazioni possa essere seguito e verificato prima di sviluppare l'applicazione
- In secondo luogo, questo modello può essere usato dagli sviluppatori per **creare il database fisico** e tutti gli oggetti che ne fanno parte

Caratteristiche del modello concettuale

- **Correttezza** (Uso corretto degli strumenti): utilizzo appropriato degli strumenti di modellazione concettuale secondo le loro finalità
- **Completezza** (Modellazione di tutti gli aspetti rilevanti della realtà): rappresentazione di tutti gli aspetti significativi del dominio applicativo da modellare
- **Chiarezza** (Modello leggibile e informazioni comprensibili): facilità di lettura del modello e comprensione delle informazioni rappresentate
- **Indipendenza** (Non dipendente dagli strumenti informatici successivi): il modello concettuale non deve dipendere da implementazioni o strumenti informatici specifici

Correttezza

Il modello concettuale deve utilizzare gli strumenti di modellazione (come diagrammi ER, UML, BPMN, ecc.) secondo la loro finalità specifica, senza forzature o interpretazioni errate.

Non si tratta solo di "usare lo strumento": un diagramma ER, ad esempio, serve a rappresentare relazioni tra entità, non flussi di processo. Usarlo per descrivere un workflow sarebbe un errore concettuale.

Coerenza semantica: Ogni simbolo (es. rombo per le relazioni, rettangoli per le entità) deve rispettare le convenzioni stabilite.

Indipendenza

Il modello concettuale deve essere astratto e neutrale, non vincolato a tecnologie, DBMS o linguaggi di programmazione specifici. Si deve descrivere *cosa fare*, non *come implementarlo*.

Vantaggi:

- Longevità (il modello sopravvive ai cambi tecnologici)
- Flessibilità (può essere implementato in diversi modi)

Esempio pratico:

- Dipendente: Scrivere nel modello "Usare PostgreSQL con trigger per gestire cancellazioni"
- Indipendente: Definire una regola di business come "Una prenotazione cancellata deve aggiornare la disponibilità della risorsa", senza menzionare tecnologie

Caratteristiche della progettazione concettuale

La progettazione concettuale deve essere:

- **Rigorosa** $\rightarrow$ per non lasciare dubbi in merito alle caratteristiche della base di dati che si sta progettando
- **Espressa con formalismi semplici** $\rightarrow$ per consentire la lettura e la comprensione anche da parte di utenti non tecnici

Gli utenti devono infatti essere certi che i progettisti abbiano compreso a fondo tutte le loro esigenze.

Il modello **entità/associazioni (relazioni)** ha queste caratteristiche e si concretizza in un documento con schemi grafici.

La progettazione concettuale → il modello concettuale

- Un aspetto rilevante del modello concettuale sono concetti e formalismi utilizzati nella costruzione del modello entità/associazioni
- Il modello concettuale, pur essendo molto utilizzato nella progettazione concettuale, **non ha una rappresentazione standardizzata**
- Il modello entità/associazioni è composto da **entità**, **associazioni** e **attributi** e che cosa si intenda con tali termini: esistono però diversi modi di rappresentarli
- La notazione classica e la notazione standard UML

Chi realizza il modello concettuale? Il database designer

Il **database designer** è responsabile dell'astrazione dei dati dal mondo reale a partire dall'analisi dei requisiti fino a ottenere la corretta modellazione degli stessi nello schema concettuale e successivamente nello schema logico.

I compiti del database designer

- Il primo compito di un database designer consiste nell'**analizzare le informazioni** raccolte durante l'analisi dei requisiti
- Il suo principale obiettivo è **costruire il modello di base**, che andrà poi raffinato e ristrutturato fino al suo completamento

Le prime operazioni che il modellatore esegue hanno il compito di classificare gli oggetti come entità oppure attributi.

Si procede partendo dalla documentazione del progetto:

- raccolta e analisi della documentazione
- definizione del glossario dei termini

Analisi della documentazione

Possiamo catalogare la documentazione utilizzata per la definizione dei requisiti:

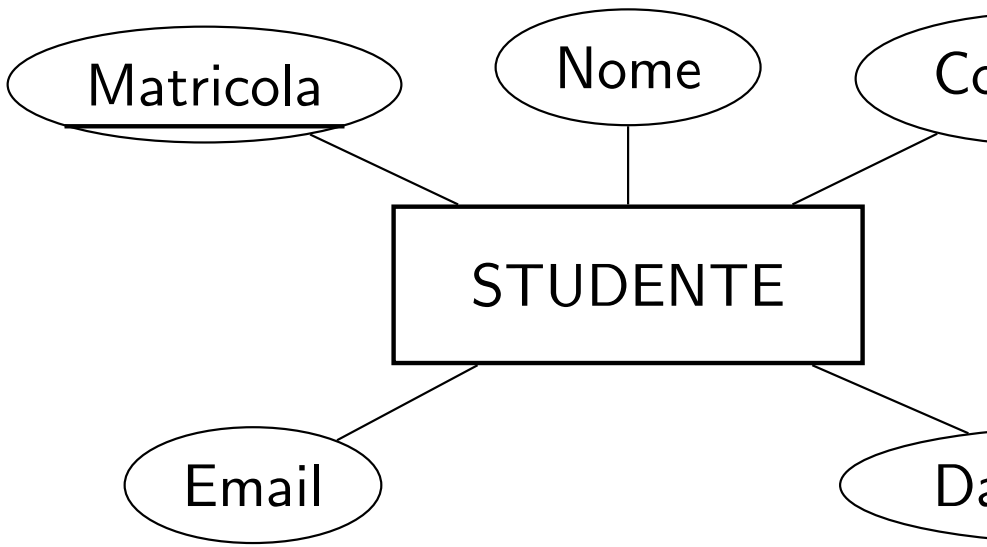
- **Documentazione specifica prodotta per il progetto**
  - le note delle riunioni tecniche e le richieste del cliente
  - gli appunti sulle interviste agli utenti finali
  - la documentazione scritta predisposta appositamente
- **Documentazione esistente**
  - le normative generali e del settore
  - i regolamenti interni
  - le procedure aziendali
- **Sistema esistente**
  - il sistema da rimpiazzare
  - le specifiche di integrazione con sistemi esistenti

Identificare gli attributi

Gli attributi descrivono un'entità: corrispondono ai campi dei record.

Regole per individuare gli attributi:

1. Gli attributi devono essere **atomici**
2. Gli attributi **derivati** non dovrebbero essere memorizzati
3. Utilizzare **codici** per classificare gli attributi



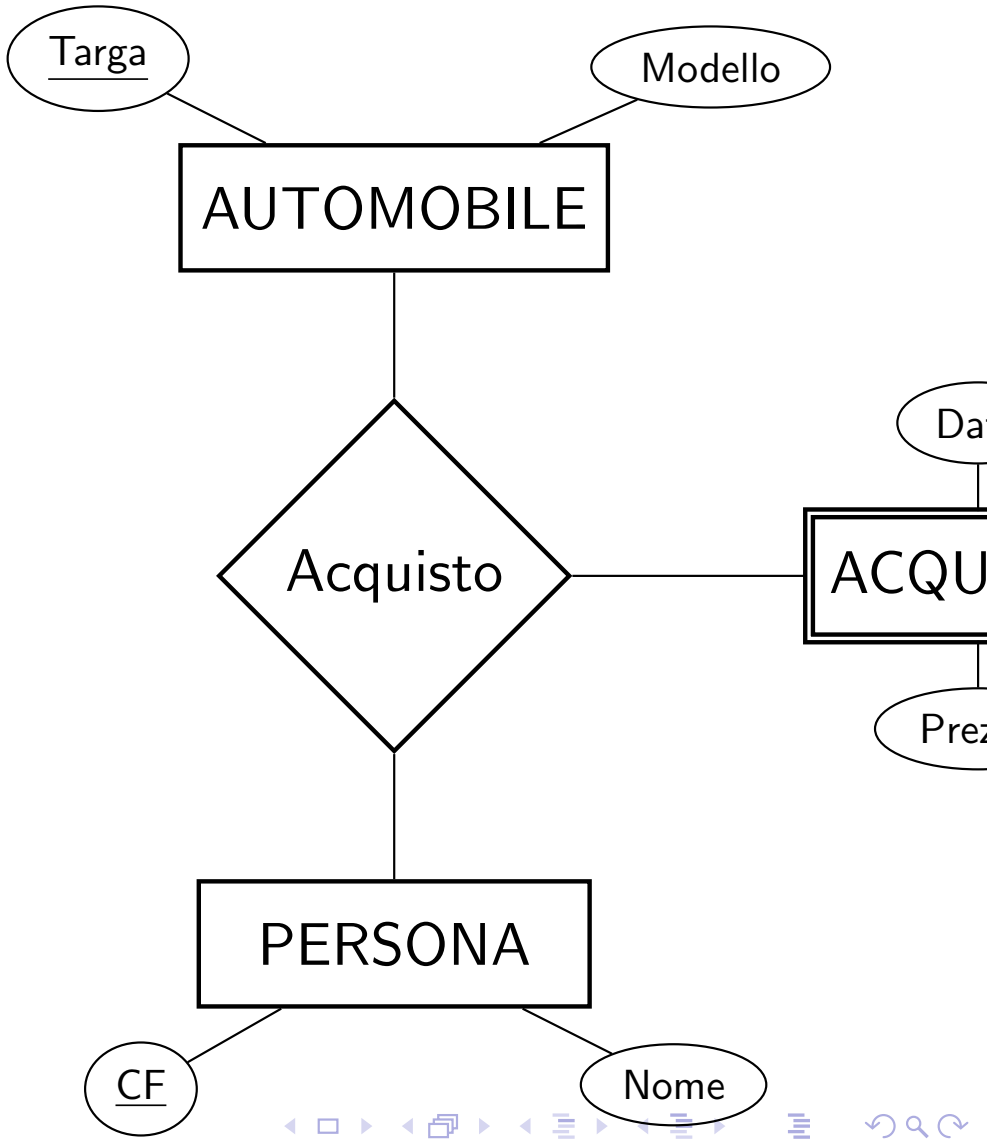
Scegliere correttamente i nomi

- I nomi degli oggetti devono essere **unici**
- Devono avere un **significato** per l'utente finale
- Devono contenere il **numero minimo di parole** di cui si ha bisogno per descrivere univocamente e accuratamente l'oggetto

Entità deboli e forti

Le entità che possiedono un insieme di attributi che le caratterizzano e hanno una **chiave primaria** prendono il nome di **entità forti**.

Esistono entità che non possiedono un proprio insieme di attributi che le identificano univocamente: tali entità si dicono **entità deboli**.



Chiavi composte

Quando nessun attributo singolo può fungere da chiave primaria, è possibile utilizzare una **chiave composta**.

Esempio: La coppia di attributi (DataAcquisto, Targa) identifica univocamente ogni singolo acquisto e la chiave primaria di Acquisto è una chiave composta, formata dalla coppia di chiavi parziali DataAcquisto e Targa (PPK, Partial Primary Key).

Alternativa: Per evitare di avere entità deboli si può aggiungere un attributo di tipo **numero progressivo** assegnato automaticamente, che identifica un'istanza dell'entità.

Molteplicità delle relazioni

La **molteplicità** di un'associazione è il numero di possibili istanze di un'entità, messo in corrispondenza con un'istanza dell'altra entità che partecipa all'associazione.

Il numero minimo e massimo di possibili istanze viene rappresentato mediante una coppia di valori separati da punti: `1..1`, `0..1`, `1..N`.

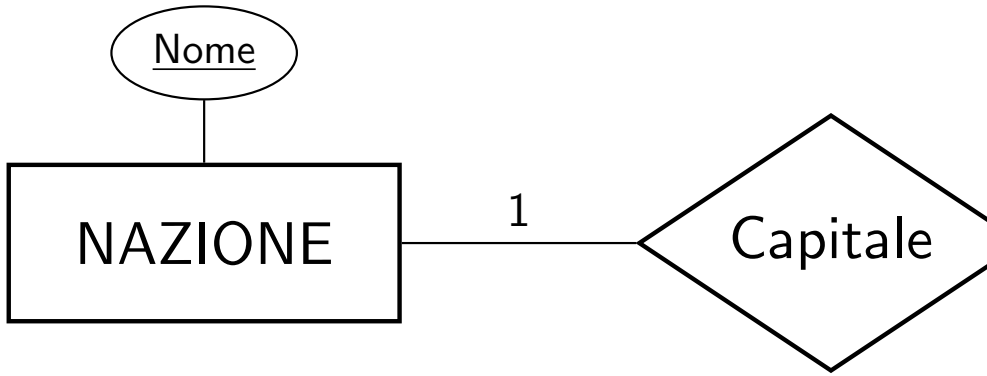
Al valore minimo e massimo sono associati gli importanti concetti di **obbligatorietà** e **cardinalità** dell'associazione:

- il valore minimo assume, in genere, uno dei due valori 0 e 1
  - 0 indica che la partecipazione è **facoltativa**
  - 1 indica che la partecipazione è **obbligatoria**
- il valore massimo definisce la **cardinalità** della partecipazione all'associazione (1 oppure N)

Le relazioni di tipo uno a uno (1:1)

Esempio: Nazione - Capitale

- Nazione ha come capitale una sola città ($1 \rightarrow 1$)
  - partecipazione obbligatoria
  - molteplicità 1
- Un capitale appartiene ad una sola nazione ($1 \leftarrow 1$)
  - partecipazione obbligatoria
  - molteplicità 1

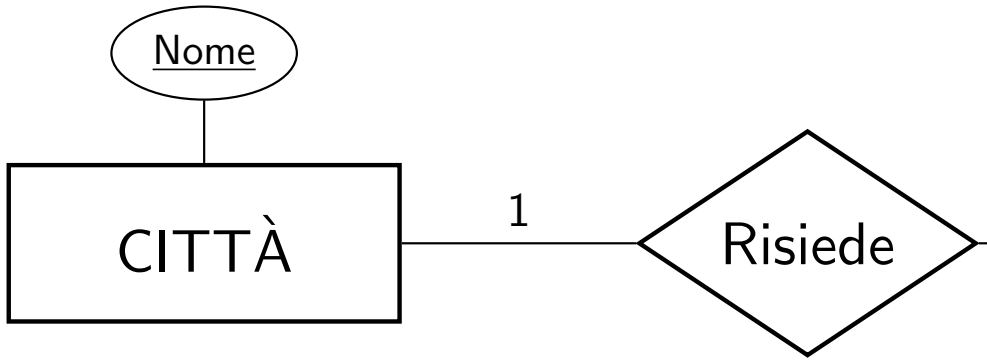


Ogni nazione ha esattamente una capitale, e ogni capitale appartiene a esattamente una nazione.

Relazioni uno a molti (1:N)

Una relazione si dice **uno-a-molti** se esiste un'istanza della prima entità cui corrisponde più di un'istanza della seconda.

Viene anche indicata con **(1, n)**.



Padre → Figlio

Esempio:

- Una persona risiede in 1 città
- In una città risiedono N persone

La città è l'entità "padre" nella relazione.

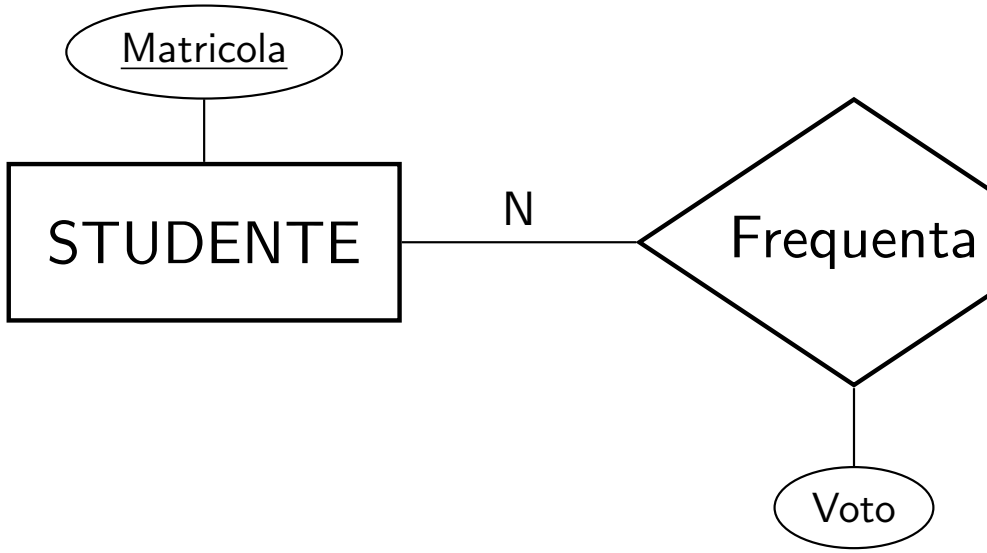
Relazioni di tipo molti a molti (N:M)

Una relazione si dice **molti-a-molti** se esiste un'istanza della prima entità in relazione con più istanze della seconda, e viceversa.

Viene indicata con **(n, n)** o **(N, M)**.

Esempio:

- Uno studente frequenta più corsi
- Un corso è frequentato da più studenti



La relazione può avere attributi propri (es. Voto).

Esistenza obbligatoria e opzionale

Esistenza obbligatoria:

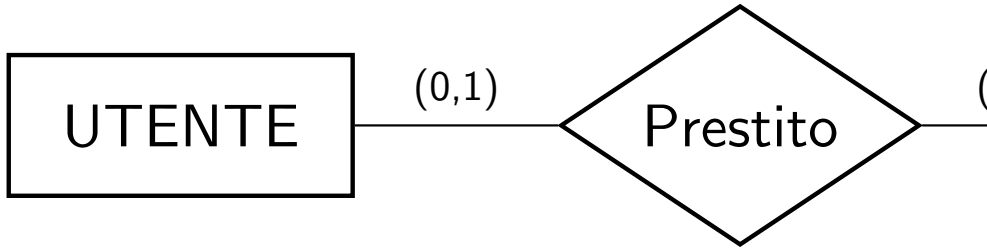
- Un'istanza deve partecipare alla relazione
- Valore minimo $= 1$

Esistenza opzionale:

- Partecipazione facoltativa
- Valore minimo $= 0$

Nel caso di relazione (1,1) il numero può essere omesso.

Esempio: Prestito libri



Un utente *può* avere in prestito *al più* 1 libro

Direzione della relazione

Riconsiderando l'esempio delle entità *studente* e *città*, si vede che, poiché la relazione tra città e studente ha cardinalità uno-a-molti, la direzione è **da città a studente** $\rightarrow$

L'entità *città* è **padre** rispetto all'entità *studente*.

La direzione della relazione indica qual è l'entità da cui parte la relazione verso l'altra entità.

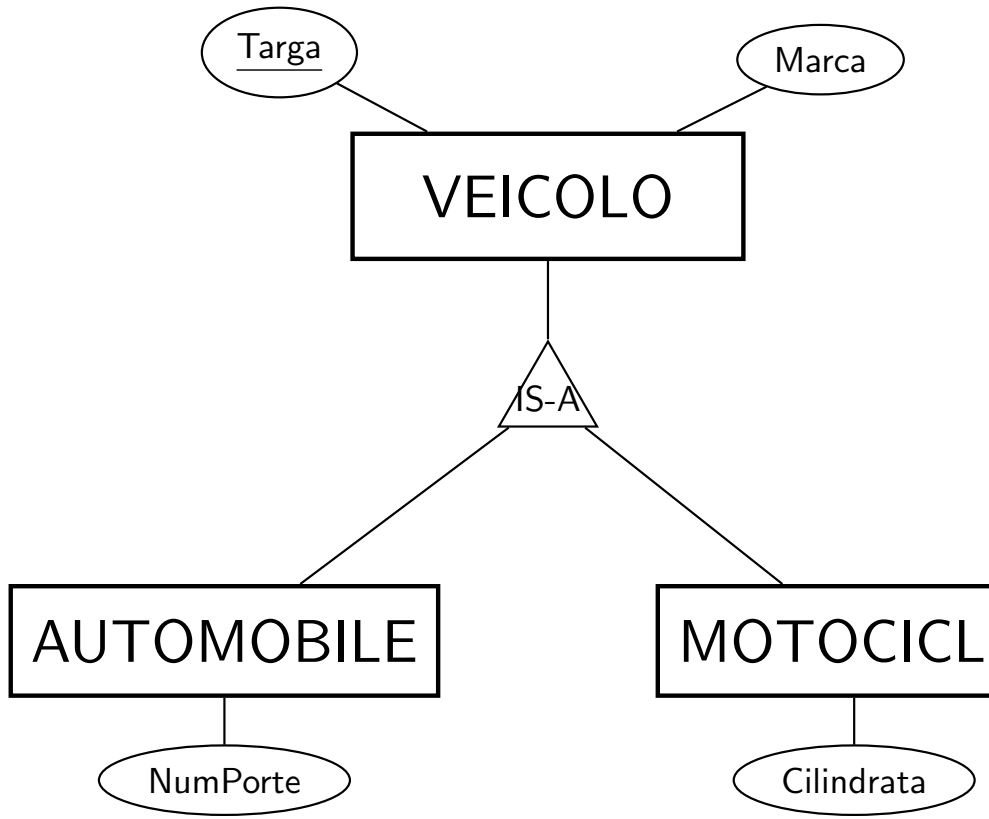
Relazione gerarchica tra entità

Esistono situazioni in cui tra le entità può essere stabilita una **gerarchia**, come nella programmazione OOP.

- **beta** è detta *sottoclasse* o *specializzazione*
- **alfa** è detta *superclasse* o *generalizzazione*

Due vincoli:

1. **Struttura**: ereditarietà di attributi e associazioni
2. **Insieme**: beta $\subseteq$ alfa

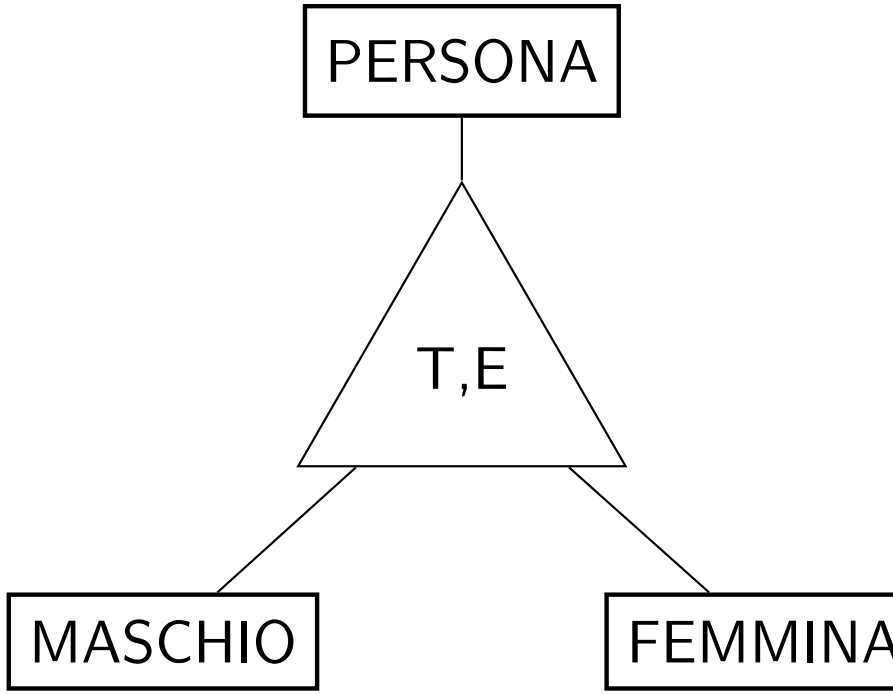


Copertura delle generalizzazioni

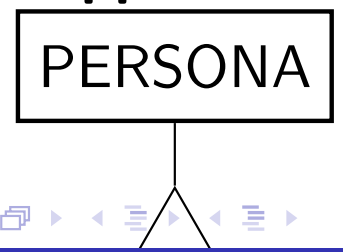
Le generalizzazioni si caratterizzano per "due dimensioni indipendenti":

1. **Unione vs classe generalizzata:**
   - **Totale**: unione $=$ generalizzata
   - **Parziale**: unione $\subset$ generalizzata
2. **Confronto fra specializzate:**
   - **Esclusiva**: disgiunte
   - **Sovrapposta**: intersezione $\neq \emptyset$

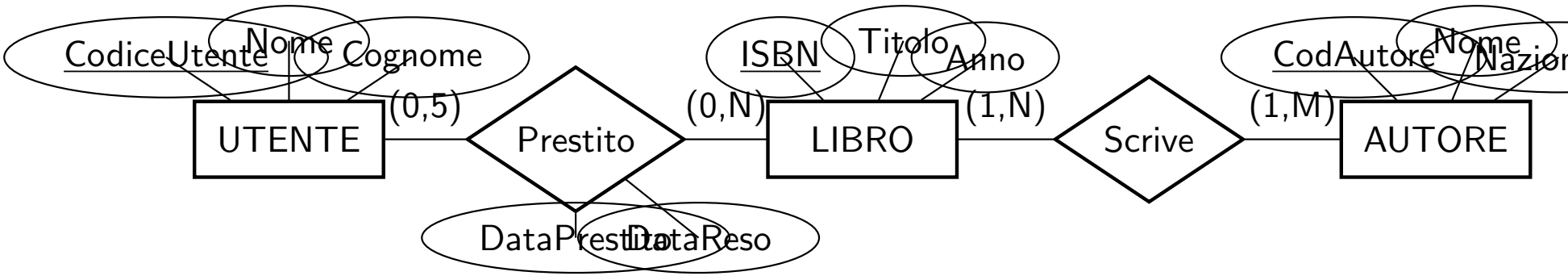
Esempio: Totale ed Esclusiva



Esempio: Parziale e Sovrapposta



Esempio completo: Sistema Biblioteca



Legenda: Un utente può avere fino a 5 prestiti, un libro può essere prestato a più utenti, un libro ha almeno un autore, un autore scrive più libri.

Grazie per l'attenzione!

Domande?