Appunti sulla Modellazione Concettuale dei Dati

Basato sulle slide del Prof. Danilo Montesi

17 maggio 2025

Indice

1 Perché la Modellazione Concettuale?

Partire direttamente a definire tabelle SQL (modello logico) è difficile e rischioso. I problemi principali sono:

- Ci si perde nei dettagli troppo presto.
- Il modello relazionale (tabelle, colonne, tipi) è troppo *rigido* per le fasi iniziali di brainstorming e analisi dei requisiti.

La soluzione è il **Modello Concettuale** (ad esempio, il diagramma Entità-Relazione - ERD):

- Permette di ragionare sulla *realtà di interesse* in modo **indipendente dall'implementazione** specifica (quale DBMS useremo, come saranno le tabelle, ecc.).
- Aiuta a definire le **classi di oggetti** (entità) e le loro **relazioni**.
- Fornisce una **rappresentazione visuale** chiara, utile per la documentazione e la comunicazione con gli stakeholder (anche non tecnici).

Esempio Pratico: Immagina di dover creare un sistema per una biblioteca. Invece di pensare subito a `CREATE TABLE Libri (...)`, con il modello concettuale pensi: "Ok, ho bisogno di `Libri`, `Utenti`, e una relazione che dice *Un Utente prende in prestito un Libro*". Questo è più astratto e flessibile.

2 Il Ciclo di Vita del Design del Database

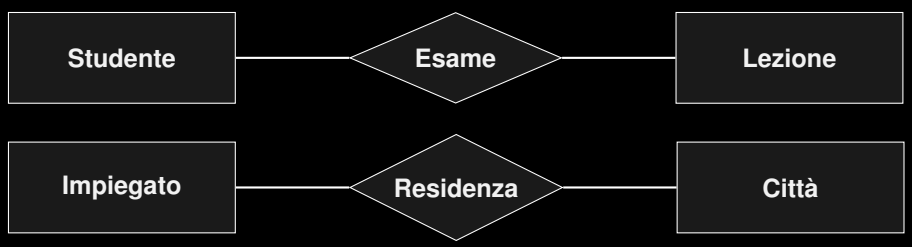
Il design del database è una fase cruciale nello sviluppo di Sistemi Informativi (SI). Le fasi principali del design sono:

2.1 Design Concettuale

- **Input:** Requisiti del database (cosa deve fare il sistema?).
- **Output: Schema Concettuale** (es. un diagramma Entità-Relazione - ERD).
- **Focus:** Il "COSA?" – quali informazioni ci servono e come sono collegate, ad alto livello.
- *Esempio Pratico:* "Abbiamo Entità `Studente` e `Corso`. Uno `Studente` si `IscriveA` un `Corso`."

2.2 Design Logico

- **Input:** Schema Concettuale.
- **Output: Schema Logico** (es. definizione di tabelle per un DB relazionale, o collezioni per un DB NoSQL come MongoDB).
- **Focus:** Il "COME?" – come traduciamo il modello concettuale in un modello supportato da un tipo di DBMS (es. relazionale, a documenti, a grafo). È indipendente dal DBMS specifico, ma non dal *tipo* di DBMS.
- *Esempio Pratico:*



- Dallo schema concettuale sopra, con un po' d'immaginazione sulle releazioni, con SQL potrebbe essere:

```sql
-- Table: Studente
CREATE TABLE Studente (
        id INT PRIMARY KEY,
        nome VARCHAR(100)
);

-- Table: Lezione
CREATE TABLE Lezione (
        id INT PRIMARY KEY,
        titolo VARCHAR(100)
);

-- Join Table: Esame (between Studente and Lezione)
CREATE TABLE Esame (
        studente_id INT,
        lezione_id INT,
        data DATE,
        voto INT,
        PRIMARY KEY (studente_id, lezione_id),
        FOREIGN KEY (studente_id) REFERENCES Studente(id),
        FOREIGN KEY (lezione_id) REFERENCES Lezione(id)
);

-- Table: Impiegato
CREATE TABLE Impiegato (
        id INT PRIMARY KEY,
        nome VARCHAR(100)
);

-- Table: Città
CREATE TABLE Città (
        id INT PRIMARY
)
```

2.3 Design Fisico

- **Input:** Schema Logico.
- **Output: Schema Fisico** (definizioni specifiche per il DBMS scelto: indici, partizionamento, filegroup, ecc.).
- **Focus:** Ottimizzazione delle performance e dello storage.
- *Esempio Pratico:* "Sulla tabella `Studenti`, creiamo un indice sulla colonna `Cognome` per velocizzare le ricerche."

3 Modelli di Dati: Costrutti, Schemi e Istanze

- **Modello di Dati:** Una collezione di "costrutti" (come i tipi di dato in programmazione) per categorizzare i dati e descrivere le operazioni su di essi.
  - Esempio: il modello relazionale usa il costrutto `relazione` (tabella) per insiemi uniformi di `tuple` (righe).
- **Schema:** La struttura invariante nel tempo dei dati (aspetto *intensionale*).
  - SQL: `CREATE TABLE Users (id INT, name VARCHAR(255));`

  - Prisma: `model User { id Int @id; name String; }`

- **Istanza:** I valori attuali dei dati in un certo momento, che cambiano nel tempo (aspetto *estensionale*).
  - SQL: Le righe effettive nella tabella `Users`: `(1, 'Alice'), (2, 'Bob')`.
  - MongoDB: I documenti effettivi nella collezione `users`.

4 Il Modello Entità-Relazione (ER Model)

È il modello concettuale più usato. Ecco i suoi costrutti principali:

4.1 Entità (Entity)

- Rappresenta una classe di "oggetti" (cose, persone, luoghi) del mondo reale che hanno proprietà comuni e un'esistenza autonoma.
- **Esempi:** `Studente`, `Prodotto`, `Dipartimento`.
- **Rappresentazione Grafica:** Rettangolo.
- **Convenzioni:** Nomi singolari, significativi.
- *Paragone Pratico:* Simile a una classe in OOP, un `model` in Prisma, o una collezione in MongoDB.

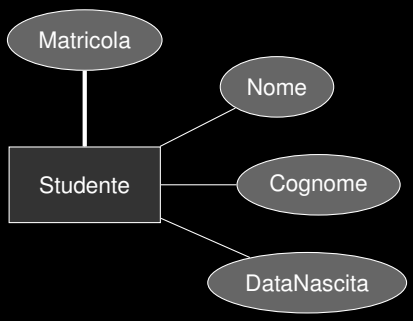


Figura 1: Esempio di entità Studente con i suoi attributi. La linea spessa indica l'identificatore (Matricola).

4.2 Relazione (Relationship)

- Un legame, un'associazione logica tra due o più tipi di entità.
- **Esempi:** `Studente` *Frequenta* `Corso`; `Impiegato` *LavoraIn* `Dipartimento`.
- **Rappresentazione Grafica:** Rombo.
- **Convenzioni:** Nomi singolari (se possibile, nomi invece di verbi).
- **Tipi:**
  - **Binarie:** Coinvolgono due entità.
  - **N-arie:** Coinvolgono più di due entità (es. `Fornitore` *Fornisce* `Prodotto` a un `Dipartimento`). Spesso si cerca di scomporle in binarie.
  - **Ricorsive:** Un'entità è in relazione con se stessa (es. `Impiegato` *Supervisiona* `Impiegato`).
    - *Paragone Pratico (Ricorsiva):* In SQL, una tabella `Impiegati` con una colonna `ID_Manager` che è una foreign key a `Impiegati.ID`.
- **Ruoli:** Utili nelle relazioni ricorsive per chiarire il significato (es. `Presidente` -(Precedente/Successivo)-> `Successione`).

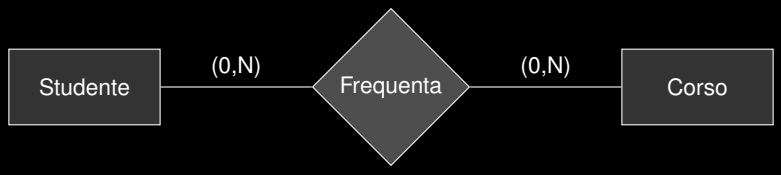


Figura 2: Esempio di relazione molti-a-molti tra Studente e Corso.

4.3 Promozione di Relazioni a Entità

Quando?

- Se una relazione ha attributi propri (es. la relazione `Iscrizione` tra `Studente` e `Corso` ha attributi come `DataIscrizione` e `VotoEsame`).
- Se uno studente può sostenere lo *stesso* esame più volte (es. per migliorare il voto). La semplice relazione `Studente-Esame-Corso` non cattura i tentativi multipli.

Come? La relazione diventa un'entità "associativa".

- *Esempio Pratico:* La relazione `Studente-Iscrizione-Corso` diventa: Entità `Studente` — Relazione `HaSostenuto` — Entità `IstanzaEsame` — Relazione `Riguarda` — Entità `Corso`. L'entità `IstanzaEsame` avrà attributi come `Data`, `Voto`.
- *SQL:* Questo si traduce in una "join table" o "tabella associativa":

```
CREATE TABLE EsamiSostenuti (
        ID_Studente INT,
        ID_Corso INT,
        Data DATE,
        Voto INT,
        PRIMARY KEY (ID_Studente, ID_Corso, Data) -- Data inclusa per tentativi multipli
);
```

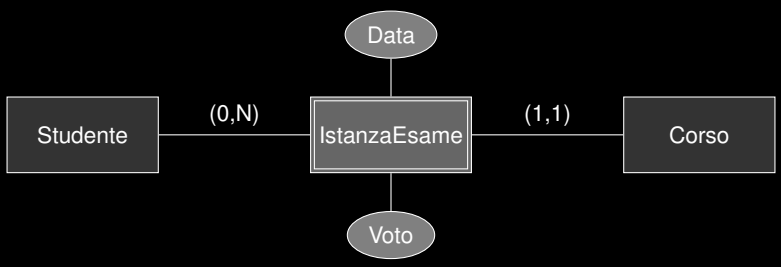


Figura 3: Esempio di promozione della relazione Esame a entità debole (doppio bordo) con attributi propri.

4.4 Attributi (Attribute)

- Una proprietà o caratteristica di un'entità o di una relazione.
- Collega ogni istanza dell'entità/relazione a un valore da un "dominio" (insieme di valori possibili).
- **Esempi:** `Nome` dell'entità `Studente`; `Data` della relazione `Esame`.
- **Rappresentazione Grafica:** Ovale.
- **Tipi:**
  - **Semplici:** Atomici (es. `Età`).

- **Composti:** Possono essere scomposti in sotto-attributi (es. `Indirizzo` composto da `Via`, `NumeroCivico`, `Città`).
  - *Paragone Pratico (Composto):* In MongoDB è naturale: `address: { street: "...", city: "..." }`. In SQL, spesso si "appiattiscono" in colonne separate (`Via`, `NumeroCivico`, `Citta`) o, se complesso, si mette in una tabella separata.

4.5 Cardinalità (Cardinality)

Specifica il numero minimo e massimo di istanze di un'entità che possono partecipare a una relazione, o il numero di valori che un attributo può assumere.

- **Notazione comune:** `(min, max)`
  - `min = 0`: partecipazione opzionale.
  - `min = 1` (o più): partecipazione obbligatoria.
  - `max = 1`: al massimo una.
  - `max = N` (o $*$): molte.

4.5.1 Cardinalità delle Relazioni

- **Esempio:** `Impiegato (1,1)` — `LavoraPer` — `(0,N) Dipartimento`
  - Un `Impiegato` deve lavorare per **esattamente un** `Dipartimento`.
  - Un `Dipartimento` può avere **da zero a molti** `Impiegati`.
- **Tipi comuni (basati su max):**
  - **Uno-a-Uno (1:1):** Es. `Persona (0,1)` — `Possiede` — `(0,1) Pacemaker`.
  - **Uno-a-Molti (1:N):** Es. `Cliente (1,1)` — `Effettua` — `(0,N) Ordine`.
  - **Molti-a-Molti (M:N):** Es. `Studente (0,N)` — `Frequenta` — `(0,N) Corso`.
    - *Paragone Pratico (M:N):* In SQL, le relazioni M:N si implementano sempre con una tabella associativa intermedia. Prisma gestisce questo in modo più astratto.

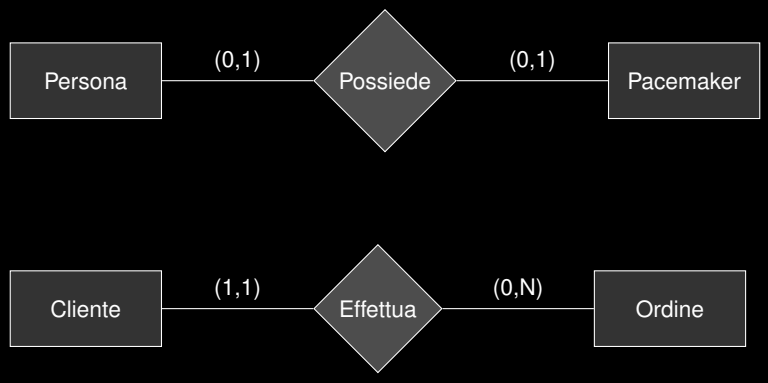


Figura 4: Esempi di relazioni uno-a-uno (Persona-Pacemaker) e uno-a-molti (Cliente-Ordine).

4.5.2 Cardinalità degli Attributi

- `(0,1)`: Attributo opzionale (può essere `NULL`). Es. `NumeroTelefonoSecondario`.
- `(1,1)`: Attributo obbligatorio, singolo valore (default). Es. `CodiceFiscale`.
- `(0,N)` o `(1,N)`: Attributo multivalore (un'entità può avere più valori per quell'attributo). Es. `NumeriTelefono` (una persona può avere più numeri).
  - *Paragone Pratico (Multivalore):* In SQL, si usa una tabella separata: `Persona(ID_Persona)`, `NumeriTelefono(ID_Persona_FK, Numero)`. In MongoDB, si usa un array: `telefoni: ["123", "456"]`.

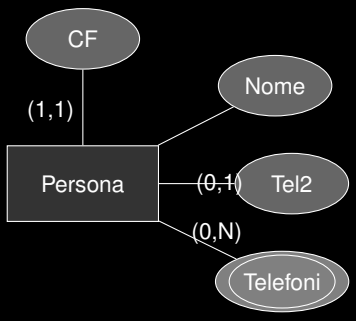


Figura 5: Esempi di attributi con diverse cardinalità: obbligatorio (CF), opzionale (Tel2), e multivalore (Telefoni).

4.6 Identificatori (Chiavi - Keys)

- Un attributo o un insieme di attributi che identificano univocamente ogni istanza di un'entità.
- **Rappresentazione Grafica:** Attributo sottolineato.
- **Tipi:**
  - **Identificatore Interno:** Formato da attributi della stessa entità.
    - Es. `codiceFiscale` per l'entità `Persona`.
    - *Paragone Pratico:* `PRIMARY KEY` in SQL; `_id` in MongoDB; `@id` in Prisma.
  - **Identificatore Esterno:** Formato da attributi dell'entità più l'identificatore di un'entità esterna a cui è collegata tramite una relazione con cardinalità `(1,1)` dal lato dell'entità da identificare. Usato per "entità deboli" che non possono esistere o essere identificate senza l'entità "forte".
    - Es. `lineId` (attributo di `OrderItem`) + `orderId` (dall'entità `Order`) identifica univocamente un `OrderItem`. `OrderItem` è un'entità debole rispetto a `Order`.
- Ogni entità deve avere almeno un identificatore.
- Le relazioni di solito non hanno identificatori (se ne hanno bisogno, si promuovono a entità).

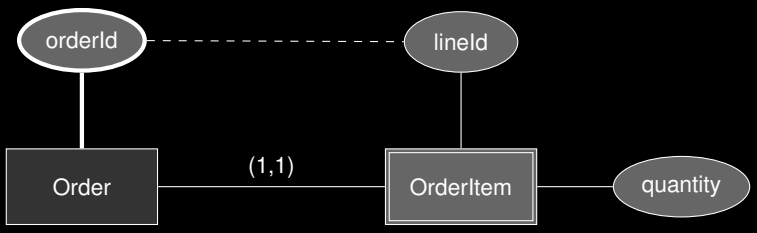


Figura 6: Esempio di entità forte (Order) con identificatore interno e entità debole (OrderItem) con identificatore esterno.

```
// Esempio con Prisma
model Order {
  orderId    Int         @id @default(autoincrement())
  // altri campi dell'ordine
  orderItems OrderItem[]
}

model OrderItem {
  orderId     Int
  lineId      Int
  quantity    Int
  order       Order  @relation(fields: [orderId], references: [orderId])
```

```
  @@id([orderId, lineId]) // Chiave primaria composta
}
```

```
-- Definizione SQL dello schema
CREATE TABLE "Order" (
  orderId    SERIAL PRIMARY KEY,
  // altri campi dell'ordine
);

CREATE TABLE OrderItem (
  orderId     INT,
  lineId      INT,
  quantity    INT,
  PRIMARY KEY (orderId, lineId),
  FOREIGN KEY (orderId) REFERENCES "Order"(orderId) ON DELETE CASCADE
);
```

4.7 Generalizzazione/Specializzazione (Inheritance)

- Una relazione tra un'entità genitore (superclasse, es. `Veicolo`) e una o più entità figlie (sottoclassi, es. `Automobile`, `Motocicletta`).
- Le figlie sono "tipi di" genitore: ereditano attributi e relazioni del genitore e possono averne di propri.
- **Rappresentazione Grafica:** Freccia (triangolo vuoto) dalle figlie al genitore.
- **Proprietà:**
  - **Ereditarietà:** Le proprietà del genitore sono implicitamente presenti nelle figlie.
  - **Copertura (Total/Partial):**
    - **Totale:** Ogni istanza del genitore DEVE essere un'istanza di (almeno) una delle figlie. (Es. `Persona` -> `Maschio`, `Femmina`).
    - **Parziale:** Un'istanza del genitore PUÒ essere un'istanza di una figlia (o solo del tipo genitore). (Es. `Veicolo` -> `Automobile`, `Motocicletta`).
  - **Disgiunzione (Disjoint/Overlapping):**
    - **Disgiunta:** Un'istanza del genitore può essere al massimo un tipo di figlia. (Es. `Persona` è `Maschio` O `Femmina`).
    - **Sovrapposta:** Un'istanza del genitore può essere più tipi di figlia contemporaneamente (raro e più complesso da modellare).
  - Di solito ci si concentra su generalizzazioni **Disgiunte (Totali o Parziali)**.
- *Paragone Pratico:*
  - OOP: `class Veicolo {}`, `class Automobile extends Veicolo {}`.
  - SQL: Ci sono diversi pattern per implementare l'ereditarietà.
  - Prisma: Può essere modellato con campi discriminatori o modelli separati con relazioni.

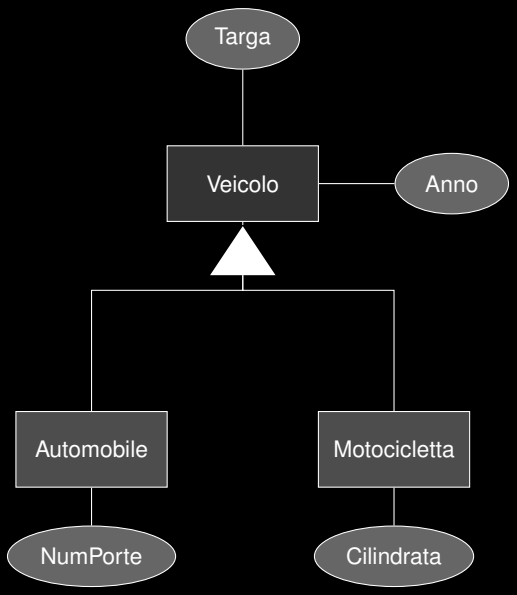


Figura 7: Esempio di generalizzazione/specializzazione: Veicolo come superclasse e Automobile/Motocicletta come sottoclassi con simbolo di ereditarietà (triangolo vuoto).

5 Documentazione

- **Dizionario dei Dati:** Descrive in dettaglio ogni entità, relazione e attributo.
- **Vincoli Non Esprimibili:** Alcuni vincoli non possono essere rappresentati graficamente nell'ERD (es. "Lo stipendio di un impiegato non può superare quello del suo manager"). Vanno documentati a parte.
  - *Paragone Pratico:* Questi vincoli si implementano spesso con `CHECK constraints` in SQL, `triggers`, o a livello applicativo.

6 UML (Unified Modeling Language) come Alternativa

- UML è un linguaggio di modellazione più ampio, usato per vari aspetti dello sviluppo software.
- Per la modellazione dei dati, si usano principalmente i **Diagrammi delle Classi (Class Diagrams)**.
- Molti concetti ER hanno un equivalente in UML:
  - **Entità** -> **Classe**
  - **Relazione** -> **Associazione**
  - **Relazione con attributi** -> **Classe di Associazione**
  - **Cardinalità:** `1`, `0..1`, `*`, `1..*`
  - **Identificatori:** `{id}` accanto all'attributo.
  - **Generalizzazione/Specializzazione:** Freccia con triangolo vuoto verso la superclasse.
  - **Concetti specifici UML:** Aggregazione (rombo vuoto), Composizione (rombo pieno).

7 Modellazione Concettuale con UML (Unified Modeling Language)

UML è un linguaggio di modellazione standardizzato e ampiamente utilizzato per specificare, visualizzare, costruire e documentare gli artefatti di un sistema software. Sebbene l'ERD sia specifico per i database, UML offre un approccio più generale e può essere utilizzato anche per la modellazione concettuale dei dati, principalmente attraverso i **Diagrammi delle Classi (Class Diagrams)**.

7.1 Classi (Classes)

In UML, un'entità del modello ER corrisponde a una **Classe**. Una classe è rappresentata come un rettangolo, tipicamente diviso in tre sezioni:

1. **Nome della Classe:** In alto, in grassetto.
2. **Attributi (Attributes):** Al centro, elencano le proprietà della classe.
3. **Operazioni (Operations/Methods):** In basso (spesso omessa nella modellazione concettuale dei dati puri, poiché ci si concentra sulla struttura).

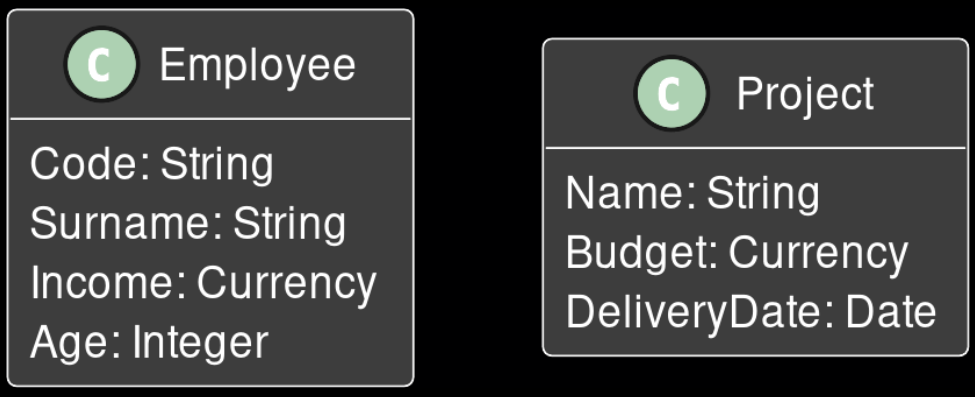


Figura 8: Esempio di Classi UML: Employee e Project.

7.2 Associazioni (Associations)

Le relazioni del modello ER sono chiamate **Associazioni** in UML. Un'associazione rappresenta una relazione semantica tra due o più classi. È disegnata come una linea continua che connette le classi.

- **Nome dell'Associazione (opzionale):** Può essere scritto vicino alla linea, spesso con una freccetta che indica la direzione di lettura (se il nome è un verbo).
- **Ruoli (opzionale):** Nomi posti alle estremità della linea di associazione per chiarire il ruolo che una classe gioca nell'associazione.
- **Molteplicità (Multiplicity):** Equivalente alla cardinalità ER, indica quante istanze di una classe possono essere collegate a un'istanza dell'altra classe. Notazioni comuni:
  - `1` (esattamente uno)
  - `0..1` (zero o uno)
  - `*` (zero o molti)
  - `1..*` (uno o molti)
  - `m..n` (da m a n)

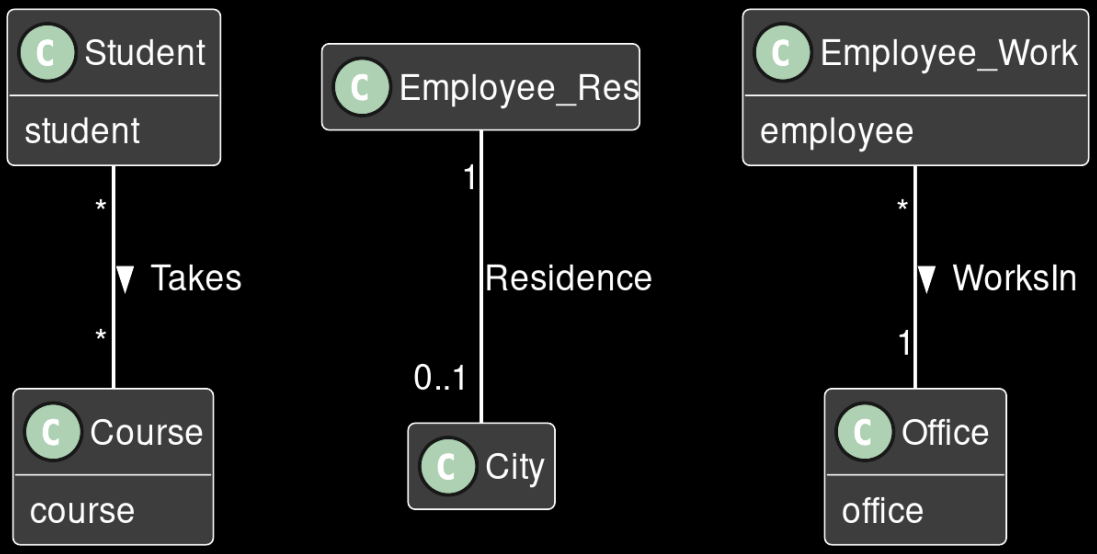


Figura 9: Esempi di Associazioni UML con nomi, ruoli e molteplicità.

7.3 Classe di Associazione (Association Class)

Quando un'associazione stessa ha attributi o operazioni, può essere modellata come una **Classe di Associazione**. È rappresentata come una classe normale collegata da una linea tratteggiata all'associazione che descrive. *Esempio:* L'associazione "Sostiene Esame" tra `Student` e `Course` può avere attributi come `Date` e `Degree`. `Exam` diventa una classe di associazione.

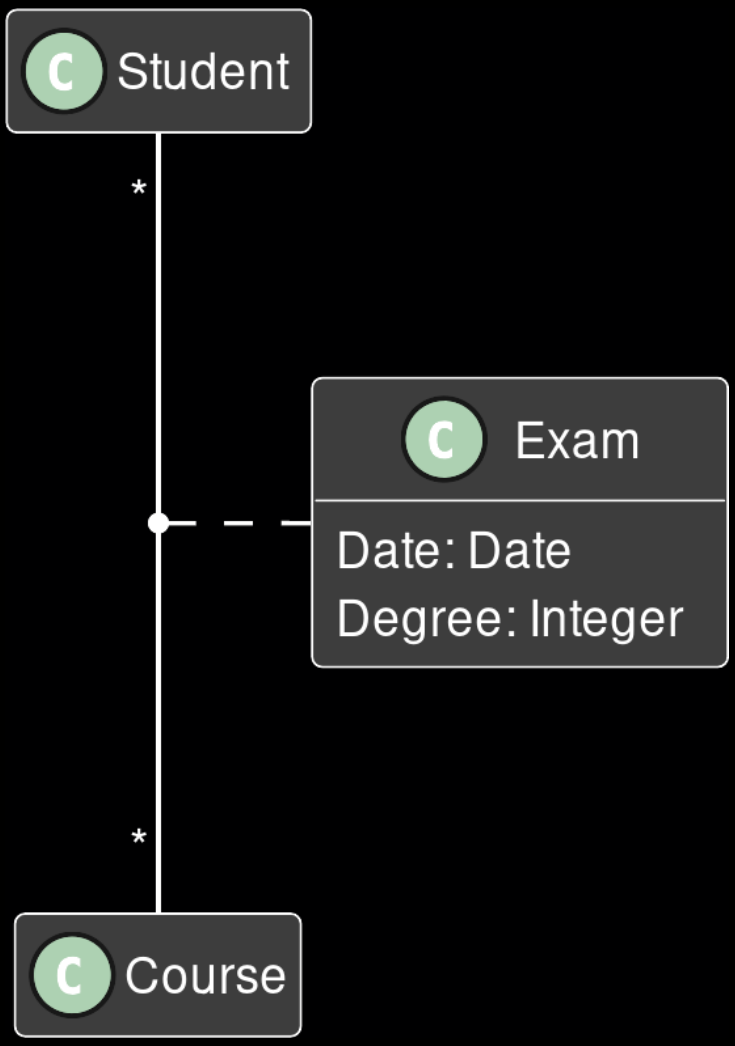


Figura 10: Esempio di Classe di Associazione UML: Exam.

7.4 Associazione N-aria (N-ary Association) e Reificazione

Un'associazione può coinvolgere più di due classi (ternaria, quaternaria, ecc.). Graficamente, si usa un rombo (come in ER) a cui sono collegate le classi. Se l'associazione n-aria ha attributi, una classe di associazione viene collegata al rombo. Le associazioni n-arie (n > 2) sono spesso complesse da gestire e interpretare. Una pratica comune è la **reificazione** (o "promozione a classe"): l'associazione n-aria viene trasformata in una nuova classe regolare, che viene poi collegata alle classi originariamente coinvolte tramite associazioni binarie.

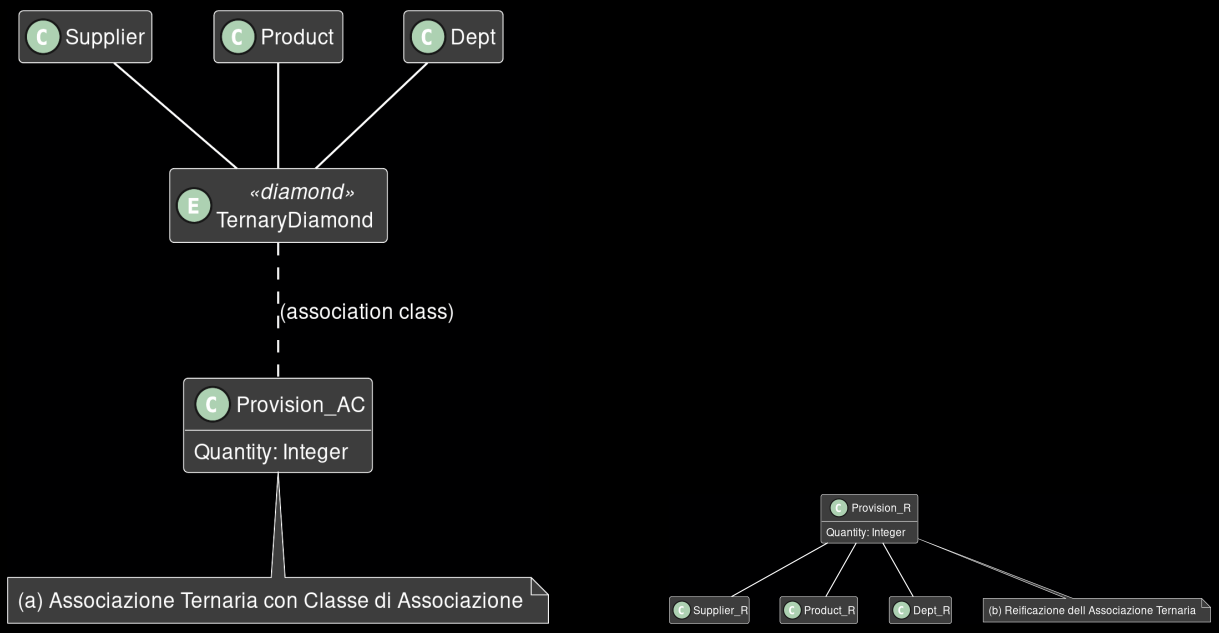


Figura 11: Associazione Ternaria con classe di associazione (a, sinistra) e sua Reificazione (b, destra) in UML.

7.5 Aggregazione e Composizione (Aggregation and Composition)

Sono tipi speciali di associazione che rappresentano relazioni "parte-di" (whole-part).

- **Aggregazione (Aggregation):** Rappresenta una relazione "ha-un" debole. Le parti possono esistere indipendentemente dal tutto. È indicata da un **rombo vuoto** dal lato del "tutto" (aggregato). *Esempio:* Un `Team` è composto da `Technician`. Un tecnico può esistere anche se il team viene sciolto, o può appartenere a più team (a seconda della molteplicità).
- **Composizione (Composition):** Rappresenta una relazione "ha-un" forte. Le parti dipendono esistenzialmente dal tutto; se il tutto viene distrutto, anche le parti lo sono. È indicata da un **rombo pieno** dal lato del "tutto" (composito). La molteplicità dal lato del composito verso la parte è solitamente `1` o `0..1` (una parte appartiene a un solo tutto). *Esempio:* Un'automobile (`Car`) è composta da un motore (`Engine`). Se l'automobile viene rottamata, anche il suo motore specifico (come parte di quell'auto) cessa di esistere in quel contesto. Una `Agency` è parte di una `Firm`.

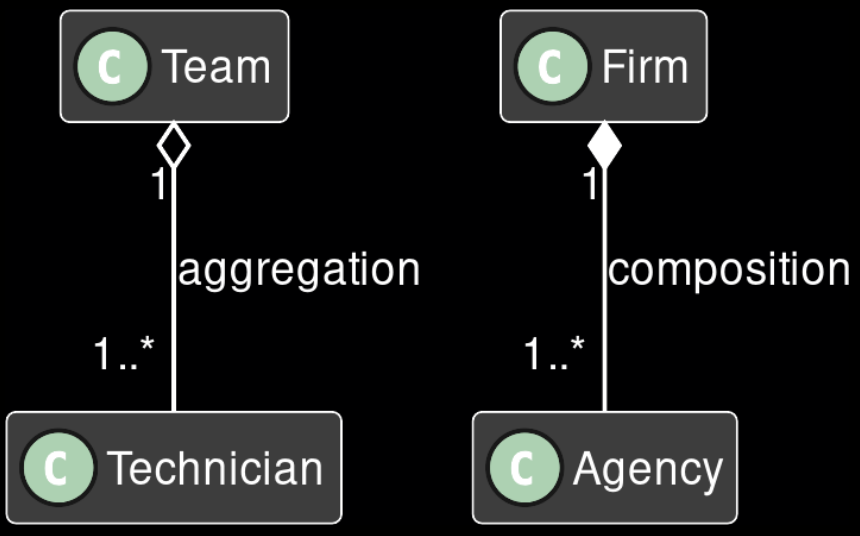


Figura 12: Esempio di Aggregazione (Team-Technician) e Composizione (Firm-Agency) in UML.

7.6 Identificatori (Identifiers)

In UML, gli attributi che compongono l'identificatore (chiave primaria) di una classe possono essere contrassegnati con la proprietà `{id}` o talvolta sottolineati (anche se `{id}` è più comune in UML2).

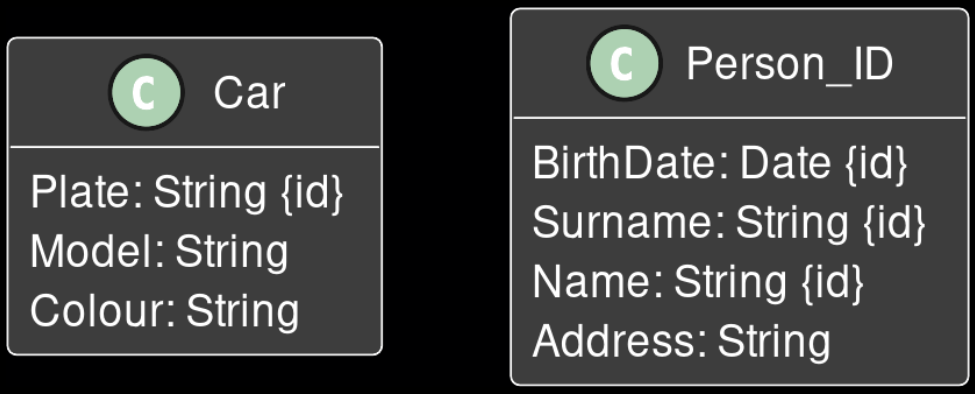


Figura 13: Esempio di Identificatori UML con `{id}`.

7.7 Identificatore Esterno (External Identifier) e Associazioni Qualificate

Un concetto simile all'identificatore esterno ER si ha quando l'identità di una classe (la "debole") dipende parzialmente da un'altra classe attraverso un'associazione. Questo può essere indicato con:

- Uno **stereotipo** sull'associazione, come `«identifying»` (come suggerito nelle slide del prof., anche se non standard UML stretto per questo specifico caso).
- Una **Associazione Qualificata**: un piccolo rettangolo (il qualificatore) è attaccato alla classe "forte", contenente un attributo della classe "debole" che, insieme all'istanza della classe forte, identifica univocamente l'istanza della classe debole.
- La molteplicità dal lato della classe "debole" verso la "forte" è spesso `1` in un'associazione identificante.

Nelle slide del Prof. Montesi (slide 100), si usa lo stereotipo `«identifiying»` (probabile typo per `«identifying»`) sull'associazione e `{id}` sugli attributi che compongono la chiave, inclusi quelli della classe "debole" e quelli ereditati implicitamente tramite l'associazione identificante.

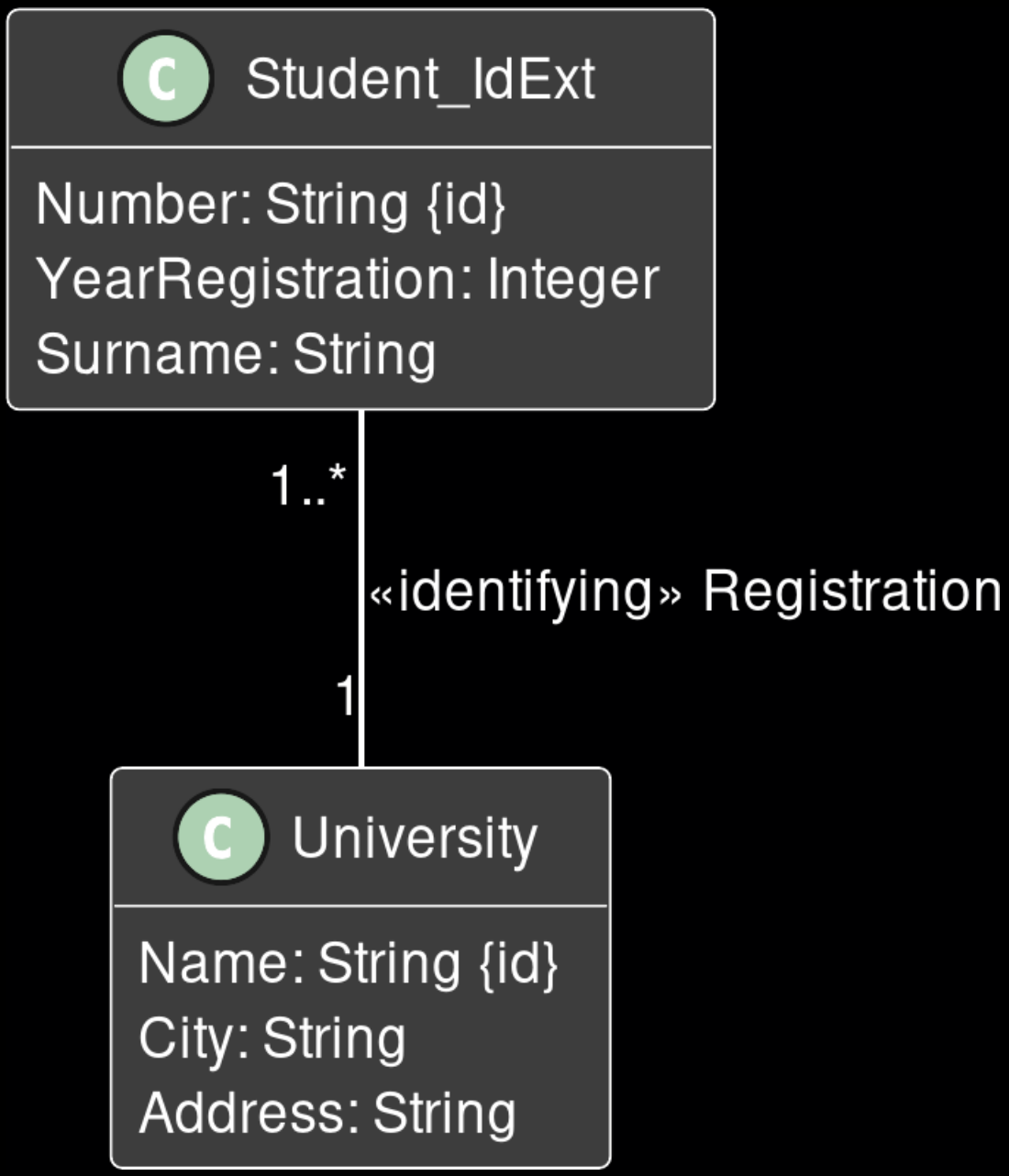


Figura 14: Esempio di associazione identificante con stereotipo (come da slide 100).

Nota: L'attributo `Number` di `Student` è `{id}` nel contesto dell'associazione con `University`. La vera chiave univoca di `Student` sarebbe una combinazione di `Student.Number` e l'identificatore di `University`.

7.8 Generalizzazione (Generalization)

Corrisponde alla generalizzazione/specializzazione del modello ER e rappresenta una relazione "è-un-tipo-di" (is-a-kind-of) tra una classe più generale (superclasse) e una classe più specifica (sottoclasse).

- **Rappresentazione Grafica:** Una linea continua con una **grande freccia triangolare vuota** che punta dalla sottoclasse alla superclasse.
- **Ereditarietà:** La sottoclasse eredita attributi, operazioni e associazioni della superclasse.
- **Vincoli:** Simili a ER, si possono specificare vincoli come:
  - `{complete, disjoint}` o `{total, disjoint}`: Ogni istanza della superclasse è esattamente una delle sottoclassi.

- `{incomplete, disjoint}` o `{partial, disjoint}`: Un'istanza della superclasse può essere una delle sottoclassi o nessuna di esse (solo la superclasse), ma non più di una.
- `{overlapping}`: Una sottoclasse può essere istanza di più sottoclassi (più raro).

Questi vincoli sono scritti vicino alla freccia di generalizzazione.

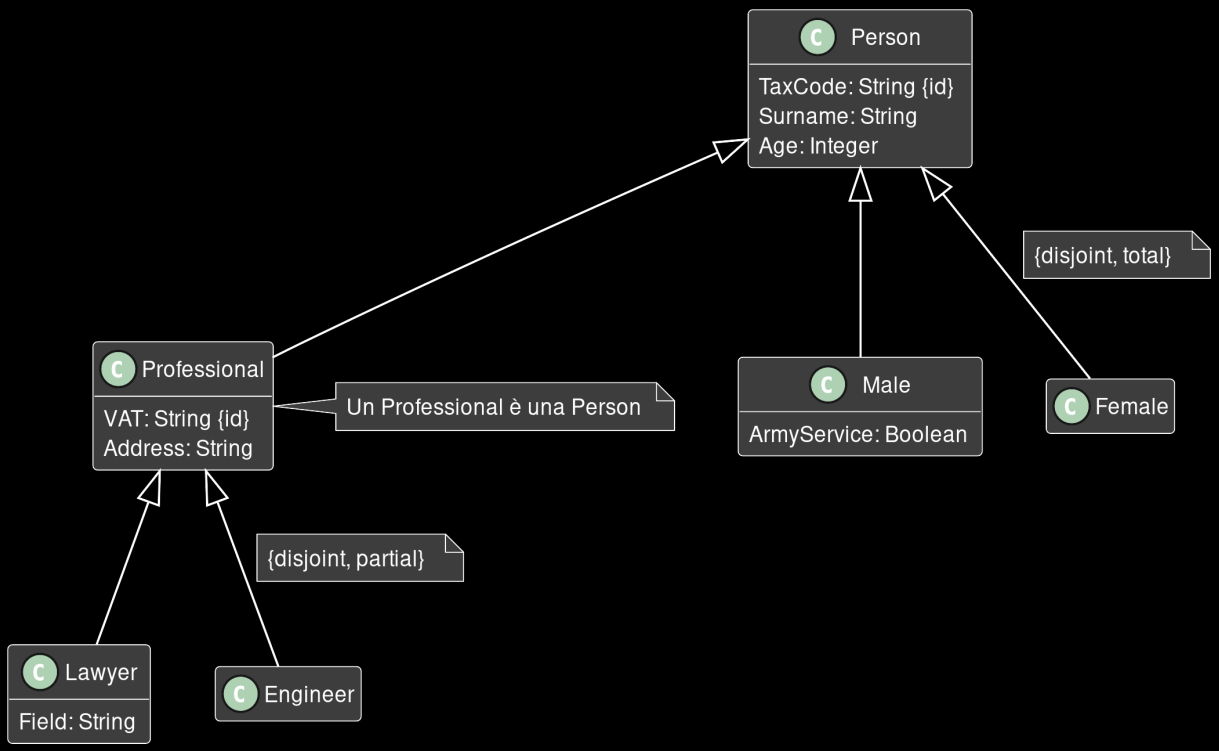


Figura 15: Esempio di Generalizzazione UML (basato su slide 101).

7.9 Esempio Complessivo: Schema Concettuale in UML

La slide 102 del Prof. Montesi mostra un diagramma ER tradotto in UML Class Diagram. Notiamo:

- Le entità diventano Classi (`Employee`, `Dept`, `Project`, `Office`).
- Le relazioni diventano Associazioni (`Management`, `Affiliation`, `Attendance`).
- `Affiliation` è una classe di associazione perché ha l'attributo `Date`.
- `Composition` tra `Dept` e `Office` è una composizione forte (rombo pieno) e identificante (`«identifying»`).
- Le cardinalità ER sono tradotte in molteplicità UML.
- Gli identificatori sono marcati con `{id}`.
- Una nota è attaccata alla classe `Project`.

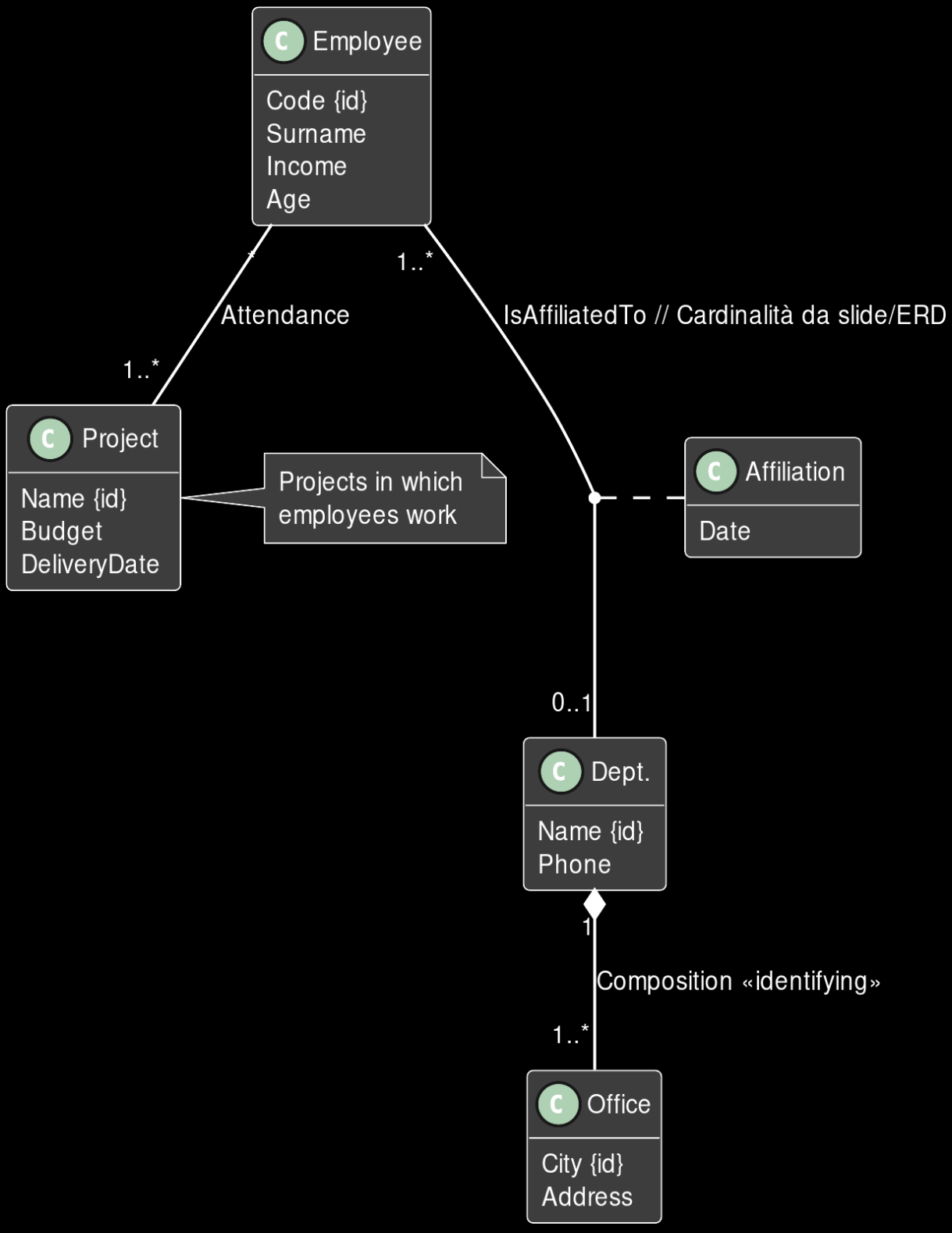


Figura 16: Diagramma concettuale UML basato sulla slide 102.

Conclusione: UML offre un set ricco di costrutti per la modellazione concettuale dei dati, con una notazione leggermente diversa ma concettualmente simile a ER. La scelta tra ER e UML Class Diagrams spesso dipende dalle convenzioni del team o dalla necessità di integrare il modello dati con altri modelli UML del sistema.