

Basi di Dati, Modulo 2

Laurea in Informatica

Parte: Cenni di ingegneria del sw
UNITÀ: B.3

Prof. Toni Mancini
Dipartimento di Informatica



SAPIENZA
UNIVERSITÀ DI ROMA

S.B.3.1

Basi di dati relazionali
La fase di progettazione
Progettazione di basi dati
relazionali



A partire dallo schema concettuale dell'applicazione (output della fase di analisi), ovvero:

- diagramma UML concettuale delle classi
- specifiche dei tipi di dato
- specifiche delle classi
- specifiche dei vincoli esterni
- diagramma UML concettuale degli use-case
- specifiche concettuali degli use-case

effettueremo le seguenti attività:

1. Decideremo la corrispondenza dei tipi di dato concettuali in domini SQL supportati dal DBMS
2. Progetteremo lo schema relazionale della base dati tenendo conto di tutti i vincoli isolati in fase di analisi concettuale e dei requisiti di performance
3. Progetteremo le specifiche realizzative delle operazioni di use-case e delle operazioni di classe.

Basi di Dati, Modulo 2

Laurea in Informatica

Parte: Cenni di ingegneria del sw
UNITÀ: B.3
Prof. Toni Mancini
Dipartimento di Informatica



SAPIENZA
UNIVERSITÀ DI ROMA

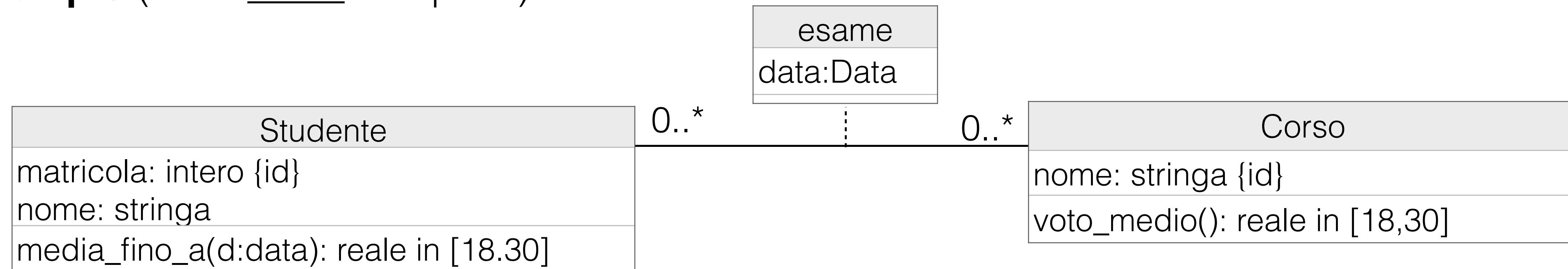
S.B.3.1

Basi di dati relazionali
Progettazione di basi dati relazionali
Produzione dello schema relazionale
con vincoli
Introduzione



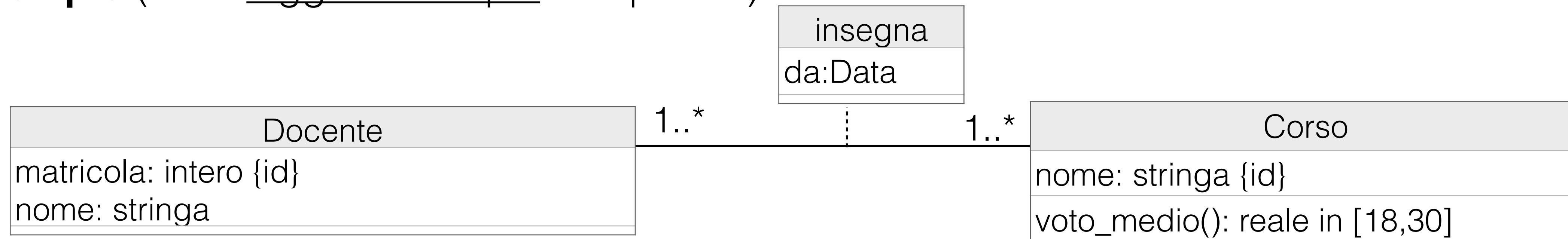
- **Obiettivo:** generare lo schema relazionale (con vincoli) per la base dati.
- **Input:** diagramma delle classi UML concettuale con classi, associazioni, attributi, is-a, generalizzazioni, vincoli di identificazione, vincoli esterni
- **Output:** un insieme di relazioni con vincoli di integrità.

- **Esempio** (caso molto semplice):



- **Il progettista decide** che le istanze di **Studente**, **Corso** ed **esame** devono essere memorizzate in un DB
- **Il progettista decide** che ogni istanza di classe o associazione sia memorizzata come una singola ennupla di una tabella
- **Il progettista definisce** il seguente schema relazionale con vincoli per il DB:
 - Tabella **Studente** (matricola:integer, nome:varchar)
 - Tabella **Corso** (nome:varchar)
 - Tabella **esame** (studente:integer, corso:varchar, data:Date)
 - foreign key: `studente` references `Studente (matricola)`
 - foreign key: `corso` references `Corso (nome)`

- **Esempio** (caso leggermente più complesso):



- **Il progettista decide** che le istanze di Docente, Corso ed insegna devono essere memorizzate in un DB
- **Il progettista decide** che ogni istanza di classe o associazione sia memorizzata come una singola ennupla di una tabella
- **Il progettista definisce** il seguente schema relazionale con vincoli per il DB:
 - Tabella **Docente** (matricola:integer, nome:varchar)
 - vincolo di inclusione: matricola occorre in insegna (docente) <— non è foreign key!
 - Tabella **Corso** (nome:varchar)
 - vincolo di inclusione: nome occorre in insegna (corso) <— non è foreign key!
 - Tabella **insegna** (docente:integer, corso:varchar, da:Date)
 - foreign key: docente references Docente (matricola)
 - foreign key: corso references Corso (nome)



- I diagrammi UML concettuali delle classi possono contenere molti costrutti ed essere complicati.
 - Come gestire attributi multivaleore?
 - Come gestire relazioni is-a, generalizzazioni disgiunte e non disgiunte, complete e non complete?
 - ...
- **Scrivere direttamente lo schema relazionale a partire dal diagramma delle classi UML concettuale comporta alto rischio di errore...**
- Vedremo una **metodologia robusta e scalabile**:
 1. **Ristrutturazione del diagramma UML delle classi**: a partire dal diagramma delle classi concettuale il progettista produce:
 - un nuovo (!) diagramma delle classi (ristrutturato). Il nuovo diagramma:
 - definisce solo le classi le cui istanze vanno memorizzate nel DB
 - non ha operazioni di classe (non supportate dai DBMS relazionali; gestite nella fase di progettazione delle funzionalità)
 - è equivalente all'originale, ovvero permette di rappresentare gli stessi livelli estensionali dei dati (al più con struttura diversa)
 - contiene solo i costrutti più semplici: classe, associazione, attributi di tipi di dato disponibili SQL (base o definiti dal progettista, v. slide su SQL) e con molteplicità massima pari ad 1
 - tiene conto di alcuni requisiti di performance
 2. **Traduzione diretta**: produciamo lo schema relazionale della base dati a partire dal diagramma delle classi ristrutturato e tenendo conto di ulteriori requisiti di performance.

Basi di Dati, Modulo 2

Laurea in Informatica

Parte: Cenni di ingegneria del sw
UNITÀ: B.3

Prof. Toni Mancini
Dipartimento di Informatica



SAPIENZA
UNIVERSITÀ DI ROMA

S.B.3.1

Basi di dati relazionali
Progettazione di basi dati relazionali
Produzione dello schema relazionale con vincoli
Ristrutturazione del diagramma UML
delle classi



- **Obiettivo:** trasformare la *porzione* del diagramma UML concettuale delle classi relativa alle classi/associazioni le cui istanze si vogliono memorizzare nel DB in un diagramma equivalente, ma dalla cui struttura si possa direttamente derivare lo schema relazionale del DB.
- **La metodologia prevede una sequenza di passi da eseguire nell'ordine dato.** Ogni passo può prevedere alcune alternative da valutare caso per caso (con un occhio alle performance):
 1. Eliminazione di attributi multivale
 2. Sostituzione dei tipi di dato concettuali con opportuni tipi supportati dal DBMS (tipi base o definiti dall'utente)
 3. Eliminazione delle generalizzazioni tra classi
 4. Eliminazione delle generalizzazioni tra associazioni
 5. Definizione di un identificatore per ogni classe
 6. Selezione di un identificatore primario per ogni classe

Il diagramma delle classi ristrutturato è, dal punto di vista concettuale, di pessima qualità.

Ma questo non è importante: la ristrutturazione è solo un modo grafico di procedere per ottenere un semi-lavorato equivalente all'originale e da cui facilmente produrre lo schema relazionale del DB!

Basi di Dati, Modulo 2

Laurea in Informatica

Parte: Cenni di ingegneria del sw
UNITÀ: B.3

Prof. Toni Mancini
Dipartimento di Informatica



SAPIENZA
UNIVERSITÀ DI ROMA

S.B.3.1

Basi di dati relazionali
Progettazione di basi dati relazionali
Produzione dello schema rel. con vincoli
Ristrutturazione del diagramma UML delle classi
Eliminazione di attributi multivaleore

Eliminazione di Attributi Multivalore

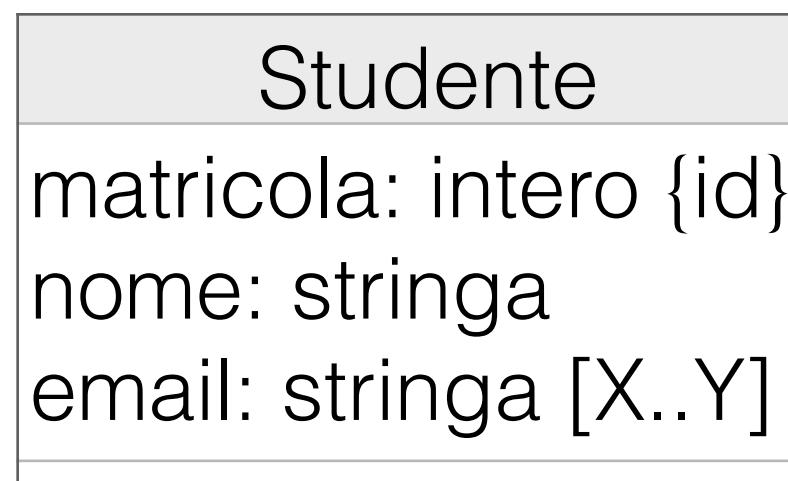
Obiettivo:

- Ristrutturare il diagramma UML concettuale delle classi in uno equivalente che **non contenga attributi multivalore** (che non sono supportati dai DBMS).

Metodologia:

- Creare una **nuova classe** le cui istanze rappresentano i valori del **tipo di dato** effettivamente utilizzate nel livello degli oggetti e dei link

Diagramma delle classi concettuale



Esito del passo di ristrutturazione



Nota: la nuova classe UML non ha alcuna rilevanza concettuale. Stiamo abusando del concetto di classe per rappresentare i valori di un tipo di dato.

Sempre 1..*!
Stiamo imponendo che il DB dovrà memorizzare solo gli indirizzi email effettivamente associati ad (almeno) uno studente

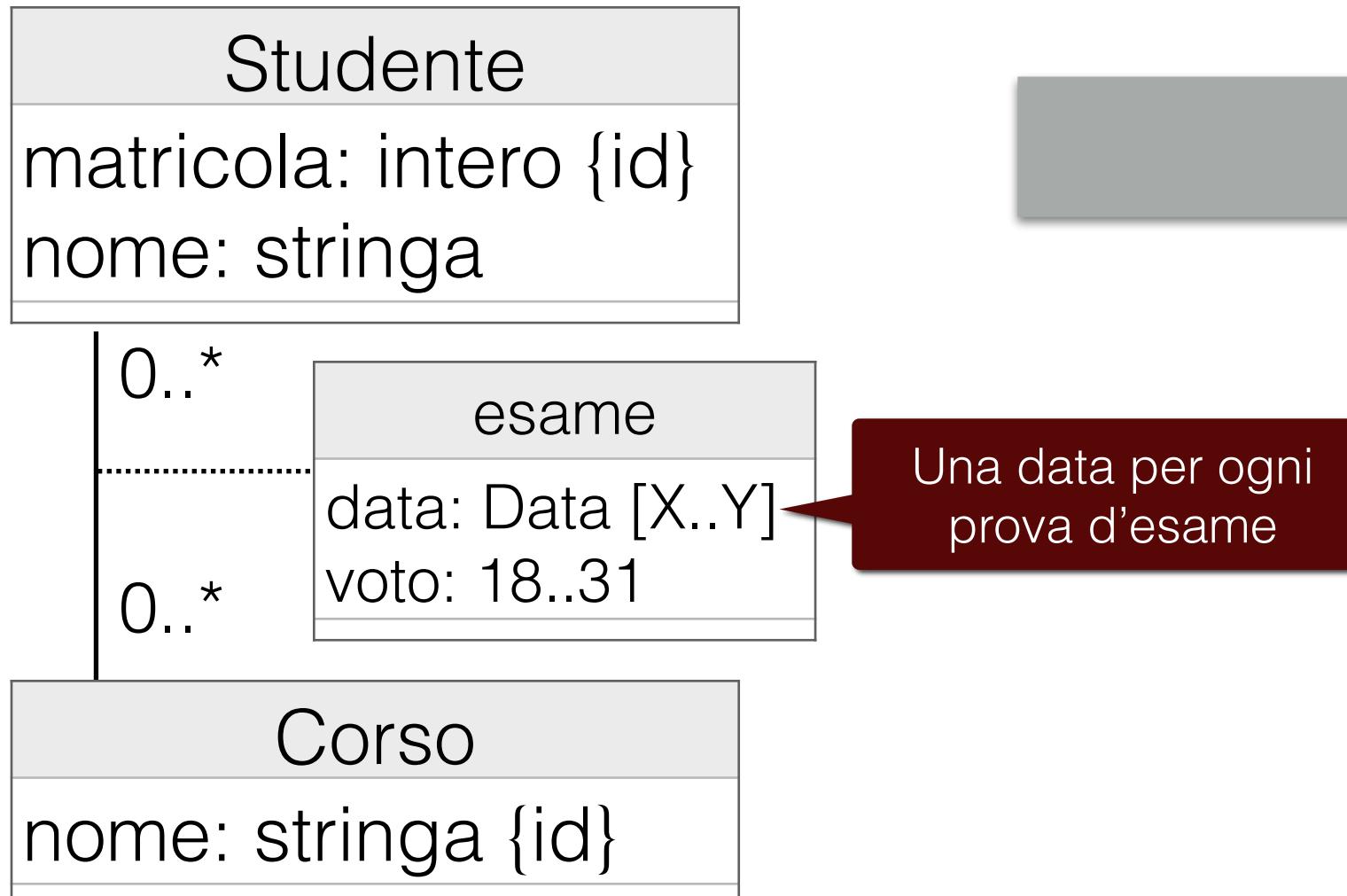
IndirizzoEmail
valore: stringa {id}

Non vogliamo rappresentare nel DB più volte la stessa istanza del tipo

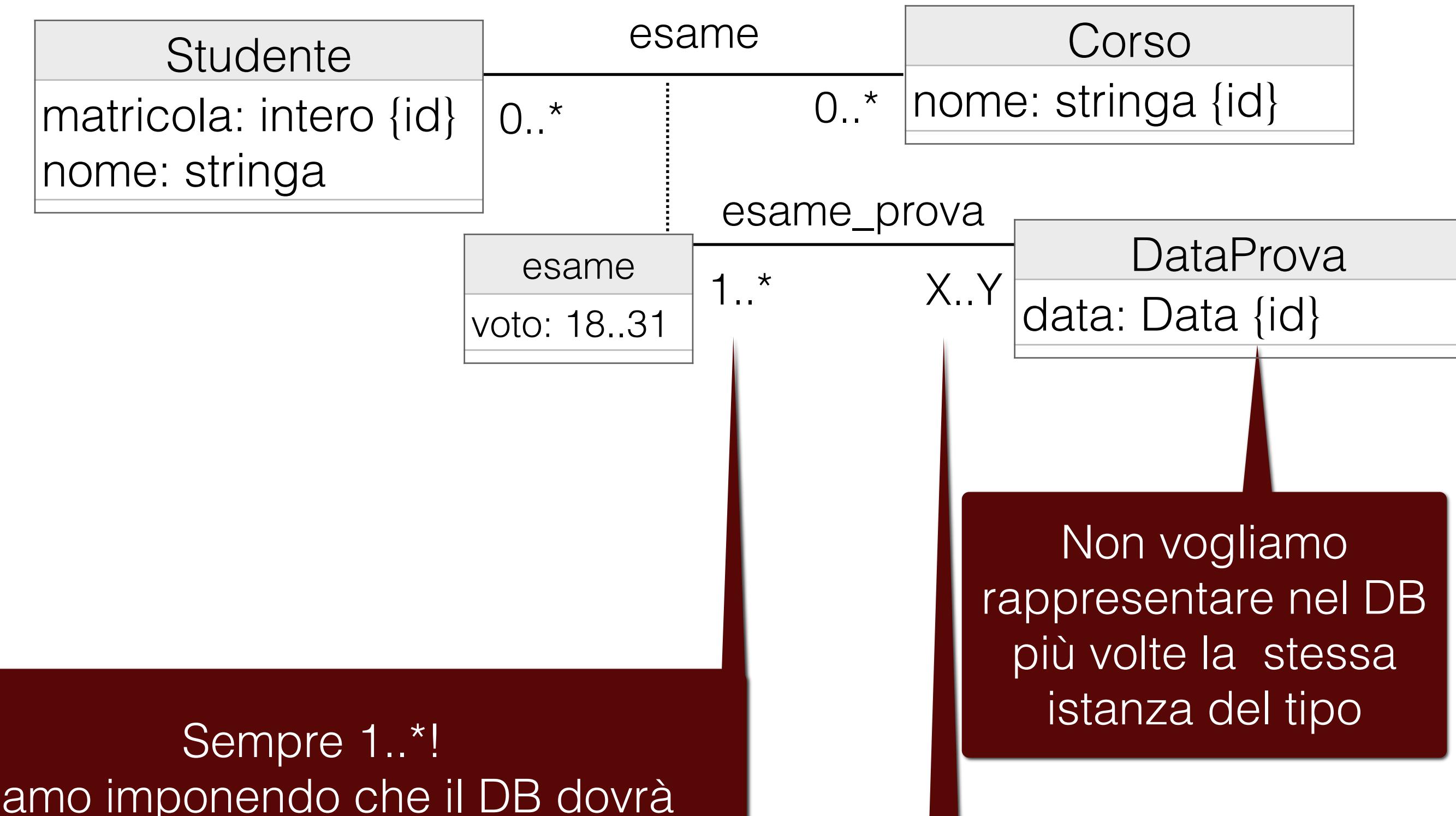
E' la molteplicità dell'attributo concettuale: X..Y

Esempio: eliminazione degli attributi multivalore di associazione

Diagramma delle classi concettuale



Esito del passo di ristrutturazione



Nota: la nuova classe UML non ha alcuna rilevanza concettuale. Stiamo abusando del concetto di classe per rappresentare i valori di un tipo di dato effettivamente utilizzati a livello estensionale.

Sempre 1..*!
Stiamo imponendo che il DB dovrà memorizzare solo le date effettivamente associate ad (almeno) un esame

E' la molteplicità dell'attributo concettuale: X..Y



Al termine del passo di eliminazione degli attributi multivalore tutti gli attributi del diagramma ristrutturato hanno vincoli di cardinalità [0..1] oppure [1..1]

Basi di Dati, Modulo 2

Laurea in Informatica

Parte: Cenni di ingegneria del sw
UNITÀ: B.3
Prof. Toni Mancini
Dipartimento di Informatica



SAPIENZA
UNIVERSITÀ DI ROMA

S.B.3.1

Basi di dati relazionali
Progettazione di basi dati relazionali
Produzione dello schema rel. con vincoli

Ristrutturazione del diagramma UML delle
classi

Scelta dei tipi di dato



Scelta dei tipi di dato

Obiettivo:

- Ristrutturare il diagramma UML concettuale delle classi in uno equivalente che contenga **solo attributi di tipi di dato (detti anche domini) supportati dal DBMS.**

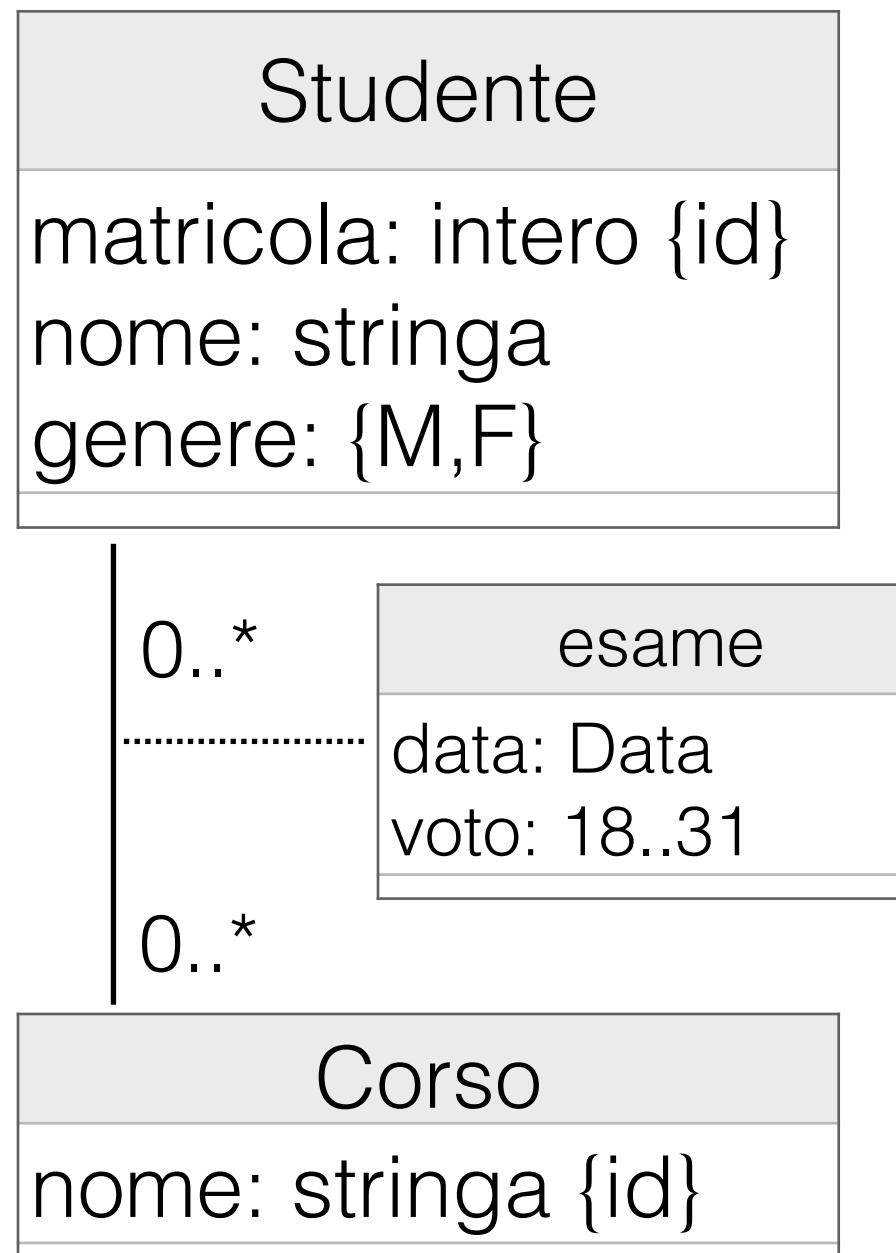
Metodologia:

- Scegliere un opportuno tipo di dato supportato dal DBMS per ogni attributo, anche definendo, tramite SQL, tipi di dato utente.
 - **Tipi concettuali base:** intero, reale, stringa, Data, Ora, DataOra
 - > hanno un immediato corrispettivo in SQL: integer, varchar(...), Date, Time, DateTime,TimeStamp, etc.
[\[link a documentazione PostgreSQL\]](#)
 - **Tipi concettuali specializzati:** ad es., intero > 0, reale <= 0, x..y, etc.
 - > definire nuovi tipi di dato supportati dal DBMS usando SQL: **CREATE DOMAIN** (v. slide su SQL DDL)
 - **Tipi concettuali enumerativi:** ad es., {M,F}, etc.
 - > definire nuovi tipi di dato supportati dal DBMS usando SQL: **CREATE TYPE ... AS ENUM** (v. slide su SQL DDL)
 - **Tipi concettuali composti:** ad es., Indirizzo, etc.
 - > definire nuovi tipi di dato supportati dal DBMS usando SQL: **CREATE TYPE ... AS (...)** (v. slide su SQL DDL)

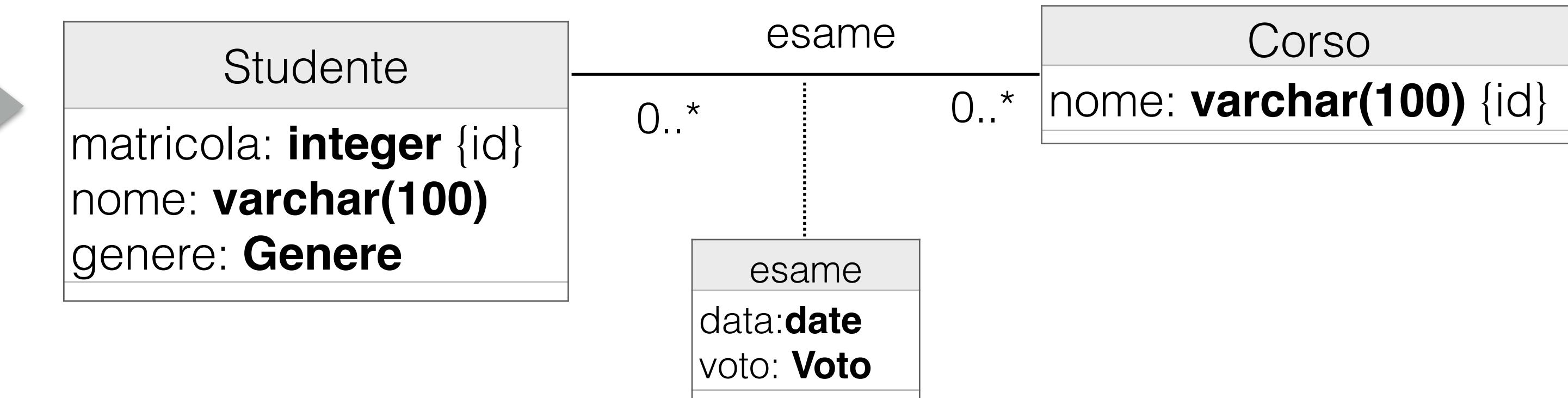
Scelta dei tipi di dato (cont.)

Esempio (tipi base, specializzati, enumerativi):

Diagramma delle classi concettuale



Esito del passo di ristrutturazione



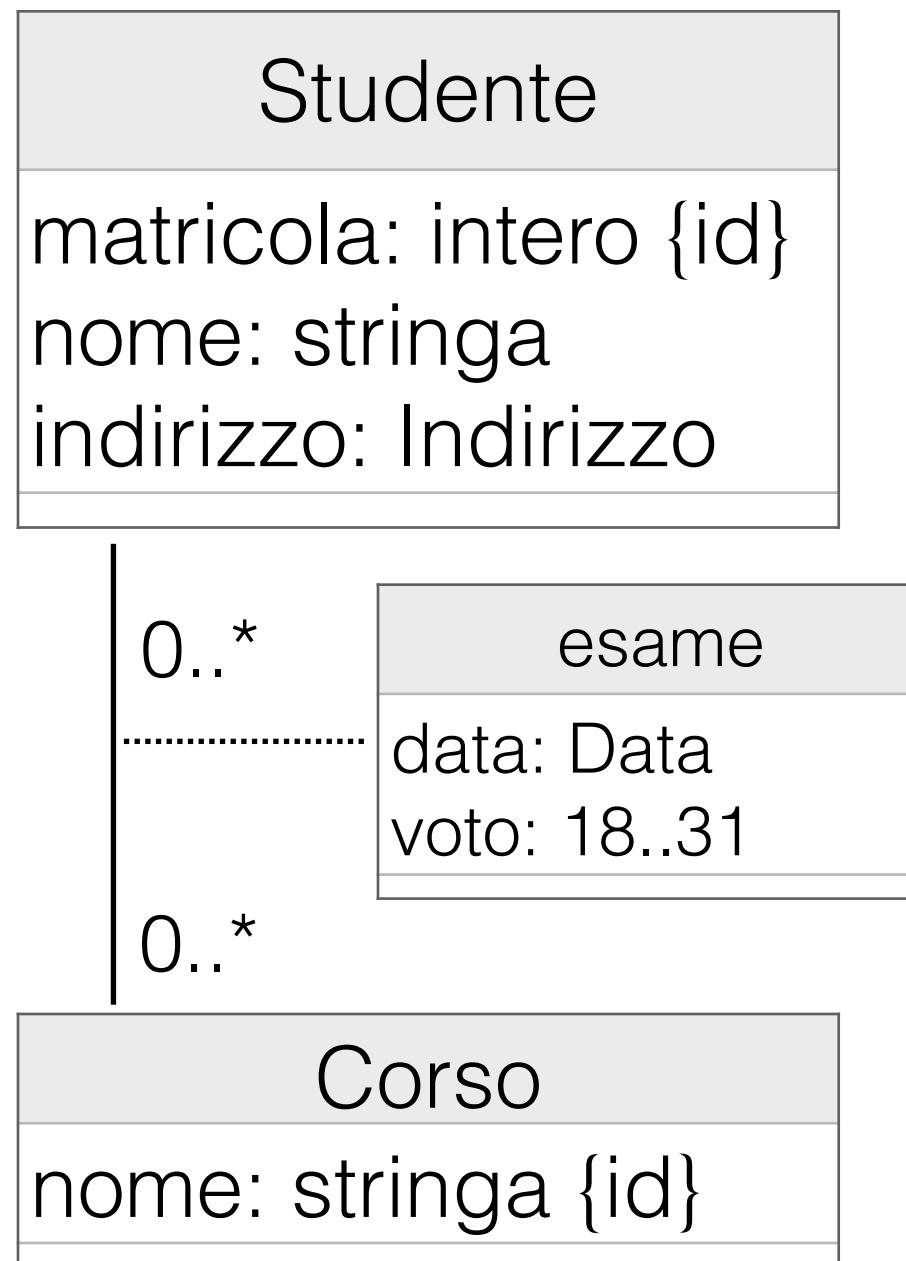
```
CREATE TYPE Genere AS ENUM ('M', 'F');
```

```
CREATE DOMAIN Voto AS Integer
CHECK (value >= 18 and value <= 31);
```

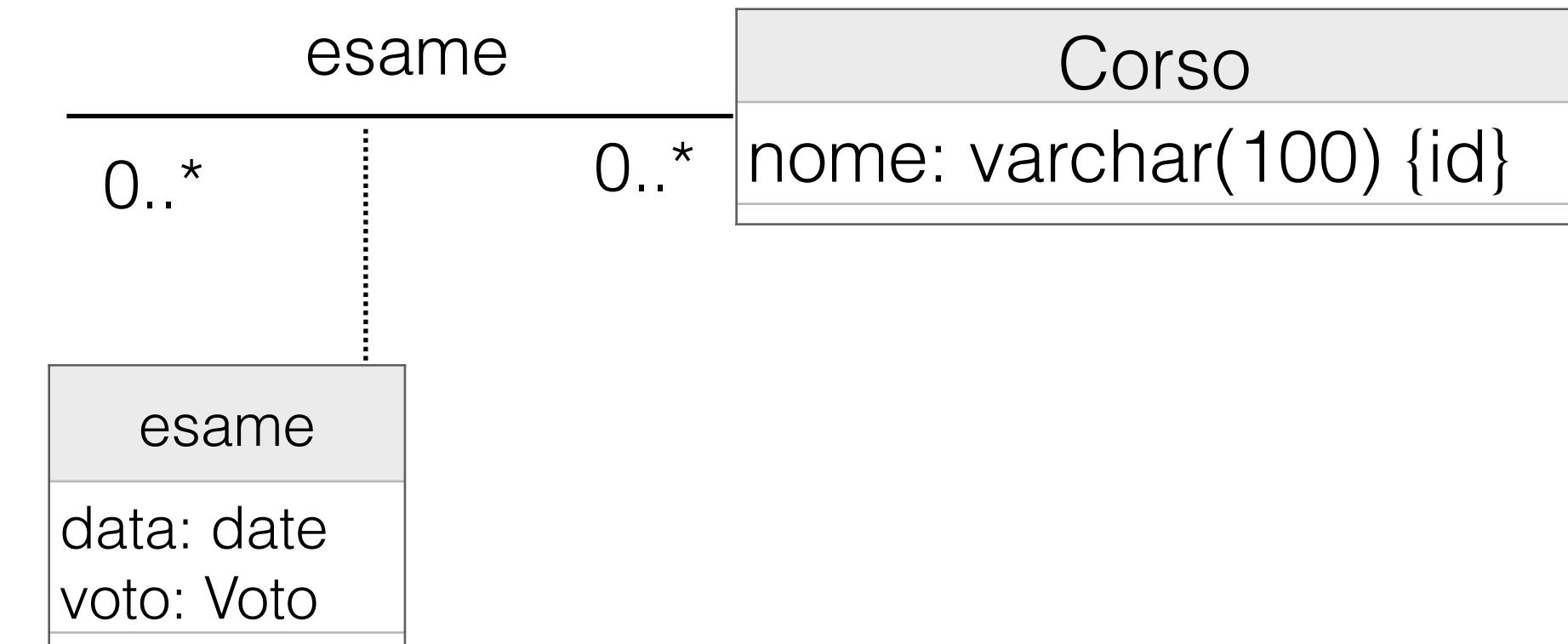
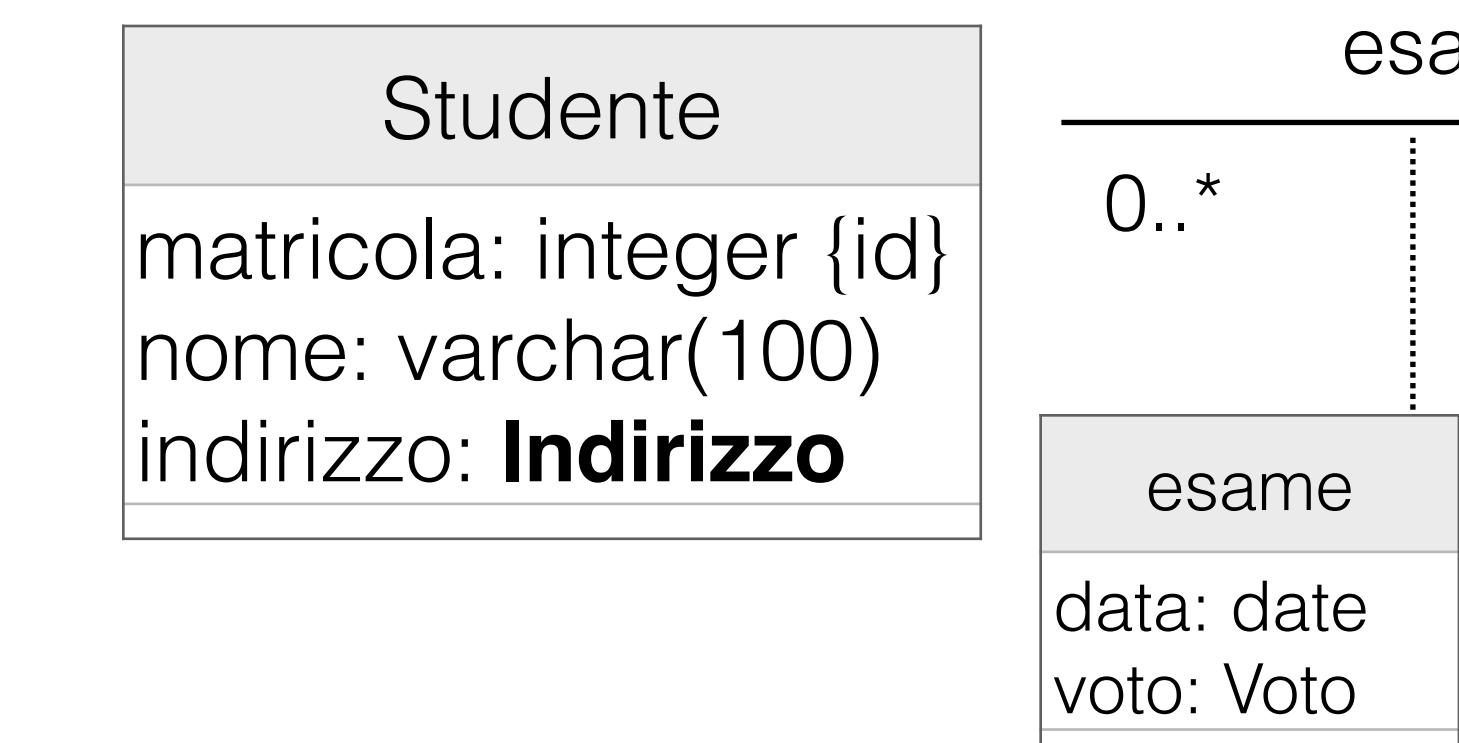
Scelta dei tipi di dato (cont.)

Esempio (tipi composti):

Diagramma delle classi concettuale



Esito del passo di ristrutturazione



```
CREATE TYPE Indirizzo (
    via varchar(100) NOT NULL,
    civico integer NOT NULL
);
```

Tipo di dato Indirizzo: record(

- via: stringa
- civico: intero

)

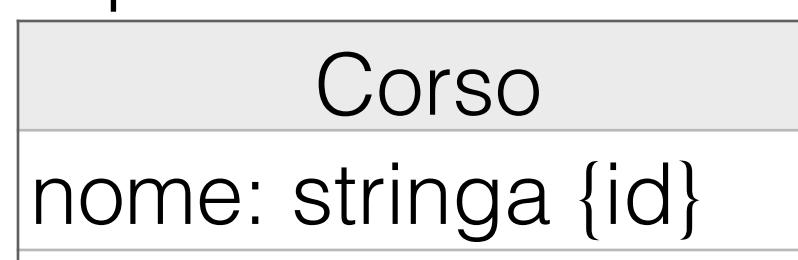
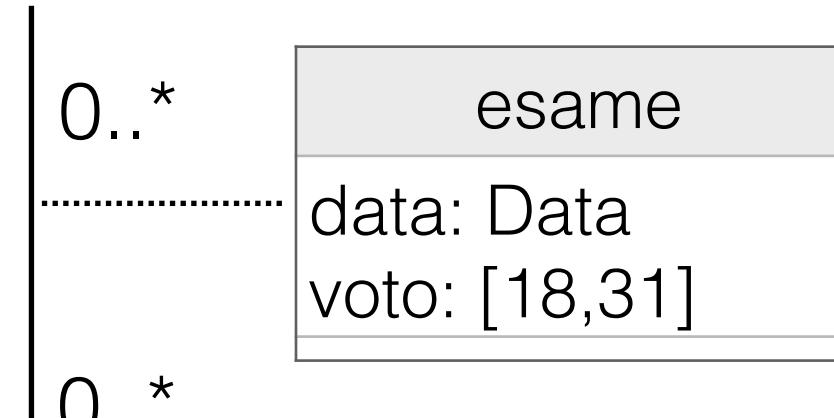
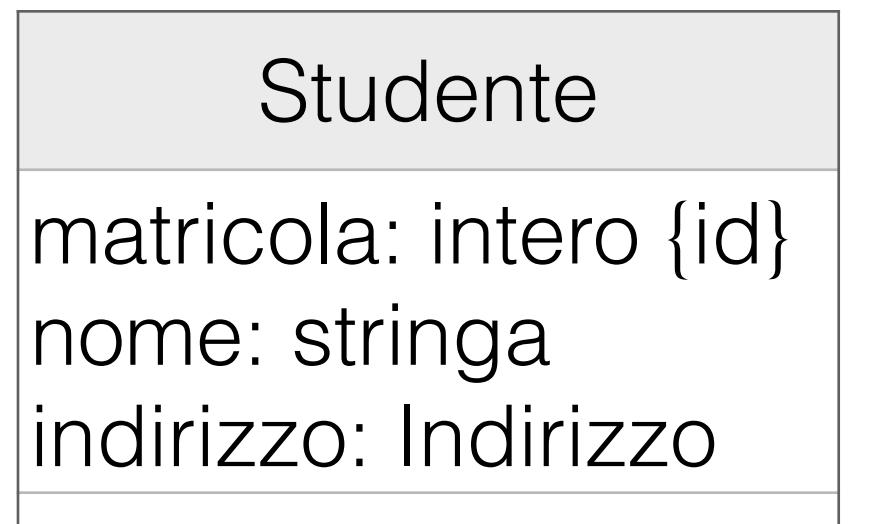
Nota: il costrutto CREATE TYPE è supportato da molti DBMS in modo limitato, v. slide su SQL DDL.

NOT NULL non supportato da PostgreSQL

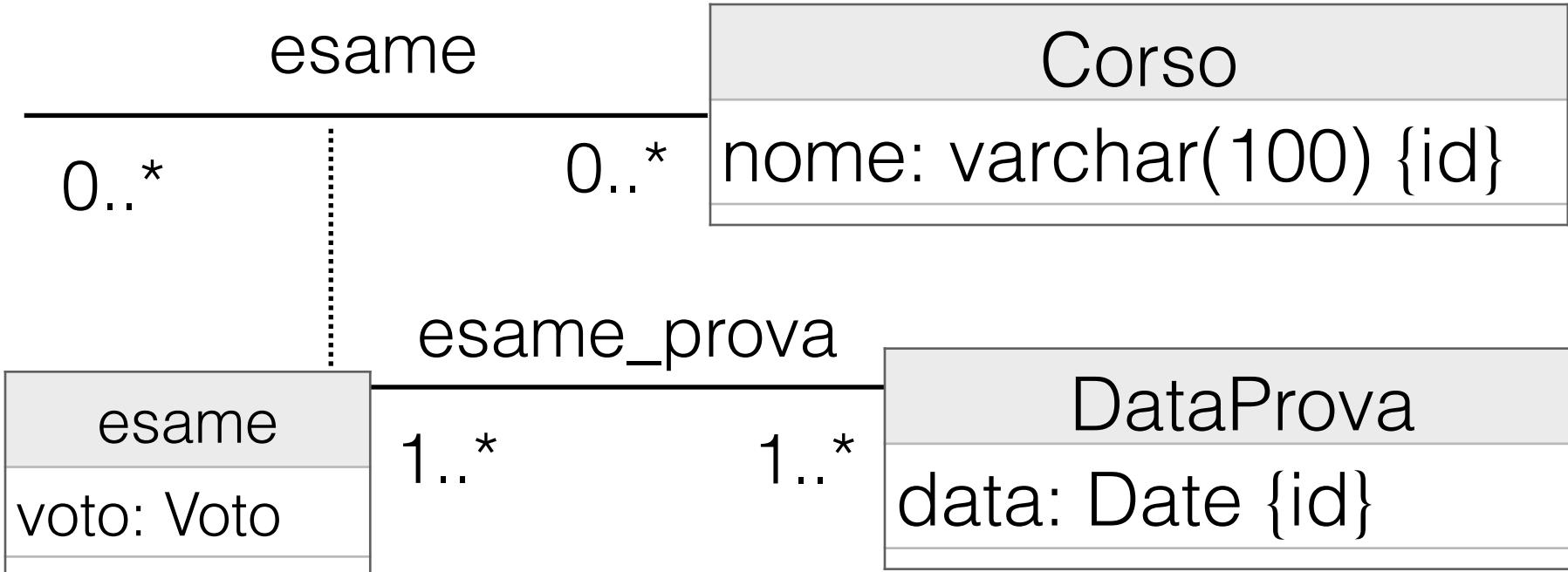
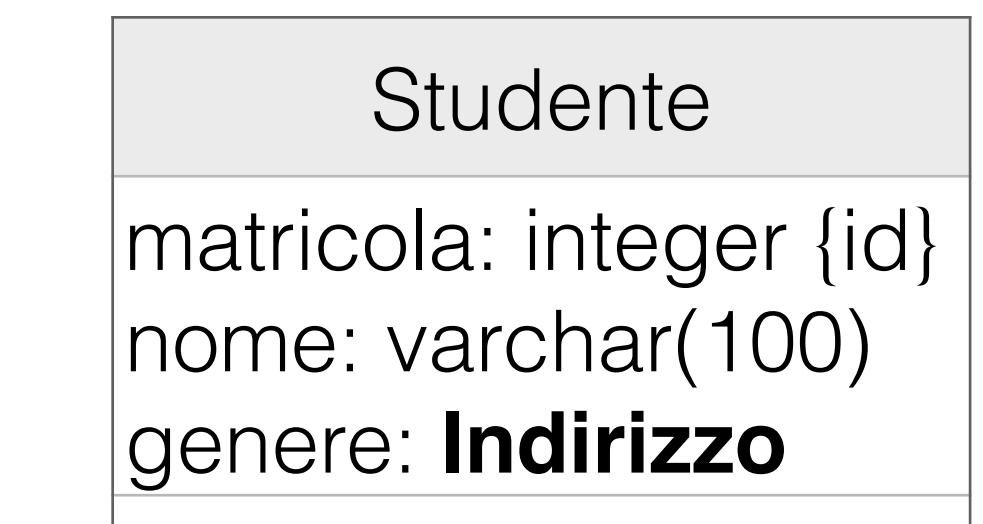
Scelta dei tipi di dato (cont.)

Esempio (tipi composti):

Diagramma delle classi concettuale



Esito del passo di ristrutturazione



```
CREATE TYPE Indirizzo (
    via varchar(100) NOT NULL,
    civico integer NOT NULL CHECK (value > 0)
);
```

Come fare?

NOT NULL e CHECK non supportati da PostgreSQL

Tipo di dato Indirizzo: record(

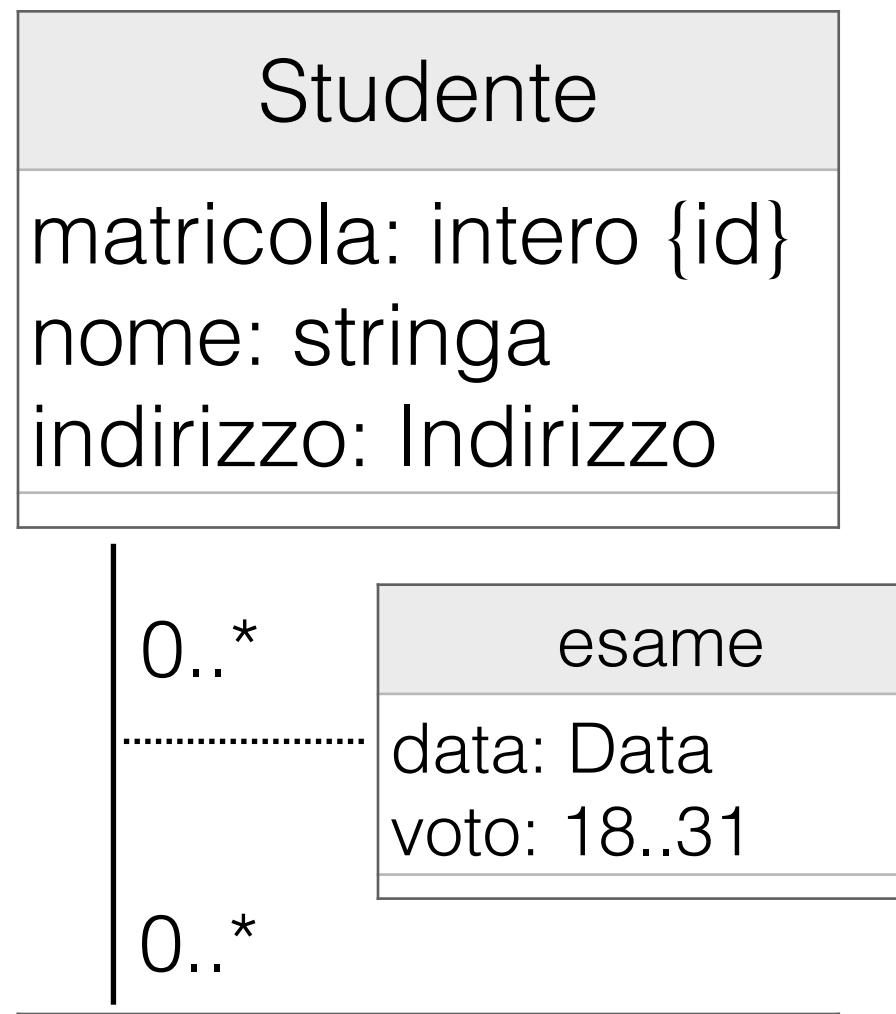
- via: stringa
- **civico: intero > 0**

)

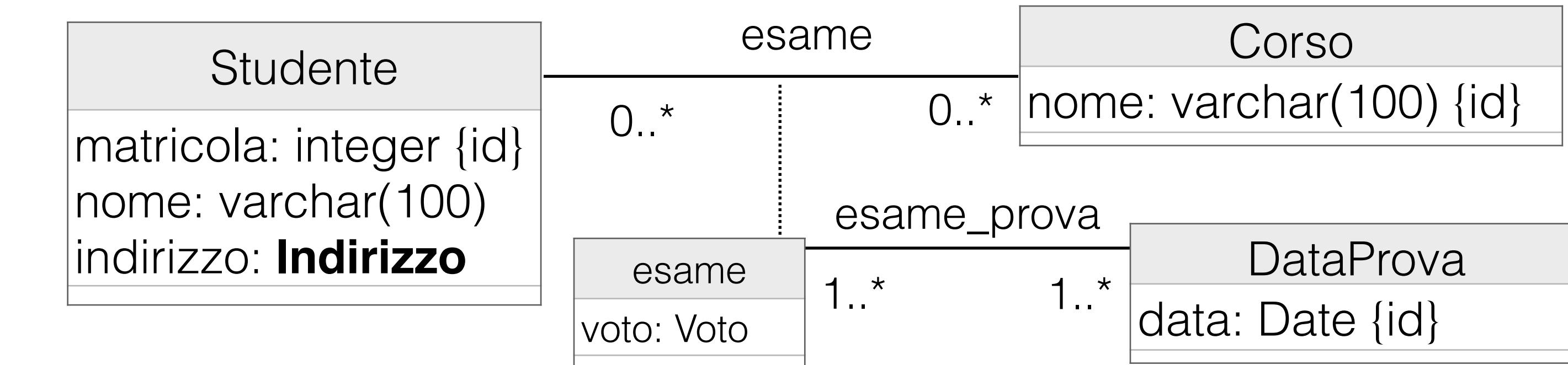
Scelta dei tipi di dato (cont.)

Esempio (tipi composti):

Diagramma delle classi concettuale



Esito del passo di ristrutturazione



Metodologia: creare domini ausiliari

```

CREATE DOMAIN String100_not_null AS varchar(100)
    CHECK (value IS NOT NULL);
    
```

```

CREATE DOMAIN Int_gz_not_null AS Integer
    CHECK (value IS NOT NULL and value > 0);
    
```

Tipo di dato Indirizzo: record(
 - via: stringa
 - **civico: intero > 0**
)

```

CREATE TYPE Indirizzo (
    via String100_not_null,
    civico Int_gz_not_null
);
    
```



Scelta dei tipi di dato (cont.)

Al termine del passo di definizione della corrispondenza dei tipi di dato concettuali in tipi supportati dal DBMS, tutti gli attributi del diagramma ristrutturato sono semplici, hanno un dominio supportato dal DBMS ed hanno vincoli di cardinalità [0..1] oppure [1..1]

Basi di Dati, Modulo 2

Laurea in Informatica

Parte: Cenni di ingegneria del sw
UNITÀ: B.3
Prof. Toni Mancini
Dipartimento di Informatica



SAPIENZA
UNIVERSITÀ DI ROMA

S.B.3.1

Basi di dati relazionali
Progettazione di basi dati relazionali
Produzione dello schema rel. con vincoli

Ristrutturazione del diagramma UML delle
classi
Generalizzazioni

Ristrutturazione delle generalizzazioni

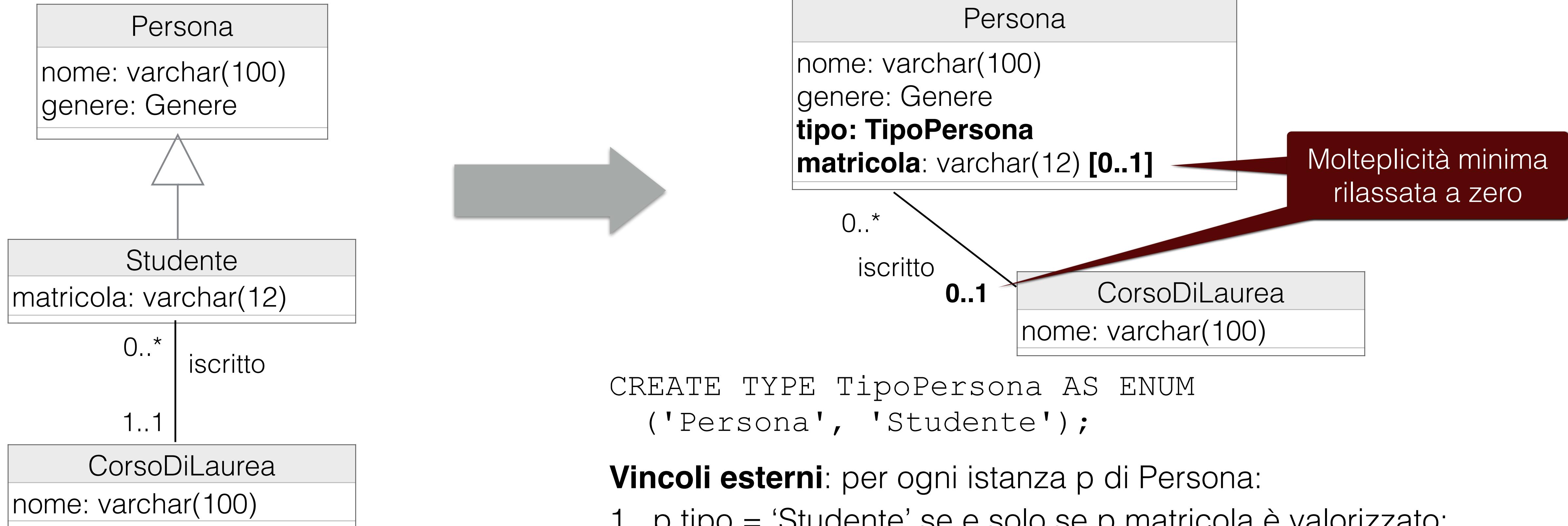
Obiettivo:

- Ristrutturare il diagramma UML concettuale delle classi in uno equivalente che **non contenga relazioni-is-a o generalizzazioni tra classi o associazioni, che non sono supportate dal DBMS.**

Metodologia:

- **Relazioni is-a o generalizzazioni tra classi:** scegliere il modo più opportuno di ristrutturare **ogni singolo livello di ogni generalizzazione**, scegliendo tra tre possibili approcci:
 - **Fusione** di sottoclassi nelle loro superclassi
 - **Divisione** delle superclassi in classi **disgiunte**
 - **Sostituzione** di relazioni is-a con associazioni
- **Relazioni is-a o generalizzazioni tra associazioni:**
 - **Fusione** di sotto-associazioni nelle loro super-associazioni, se opportuno a valle del passo precedente
 - Aggiunta di **vincoli esterni** (che saranno opportunamente gestiti più avanti)

Approccio: fondere un intero livello della generalizzazione (superclasse + sottoclassi) in un'unica classe

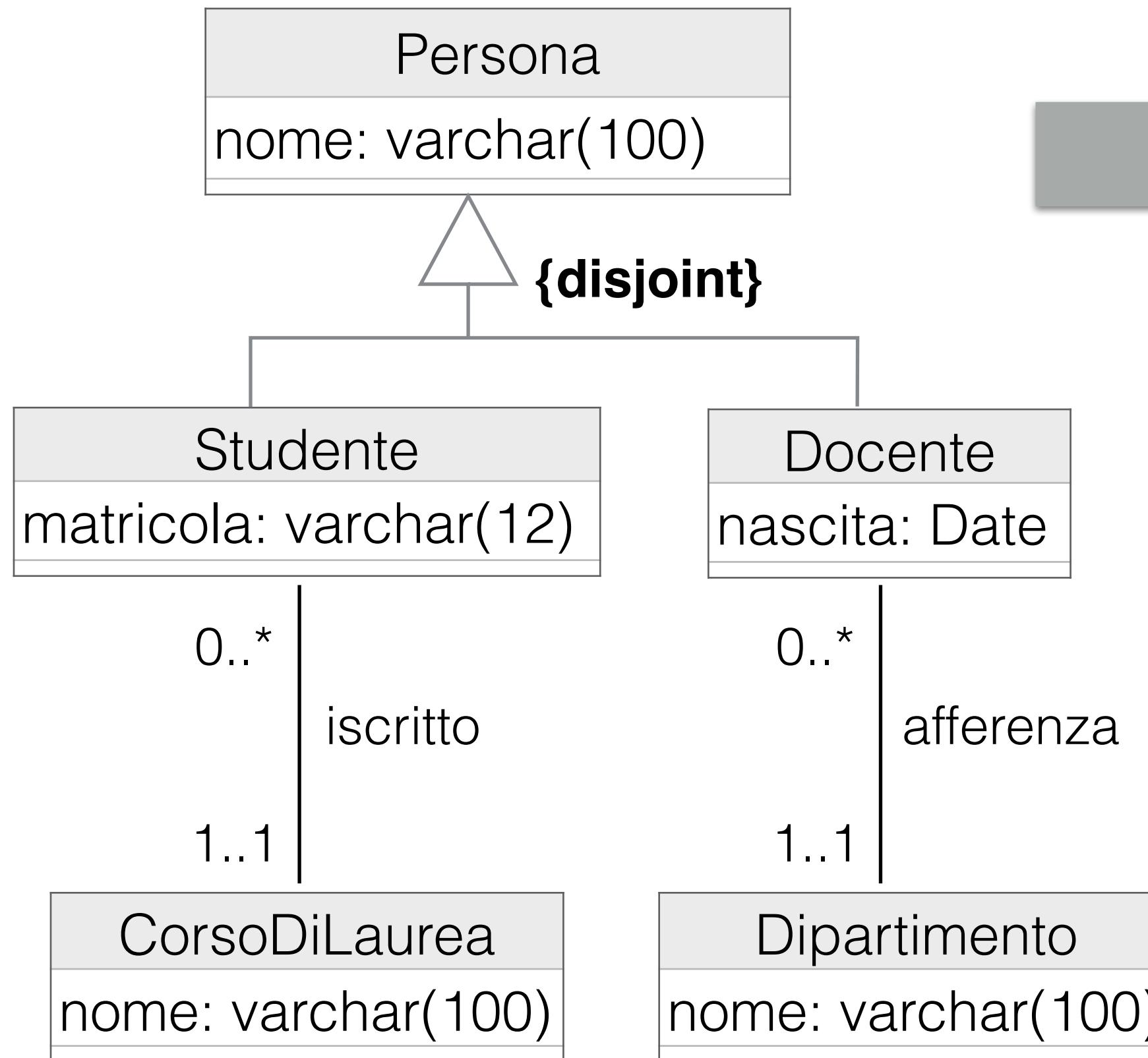


CREATE TYPE TipoPersona AS ENUM
('Persona' , 'Studente');

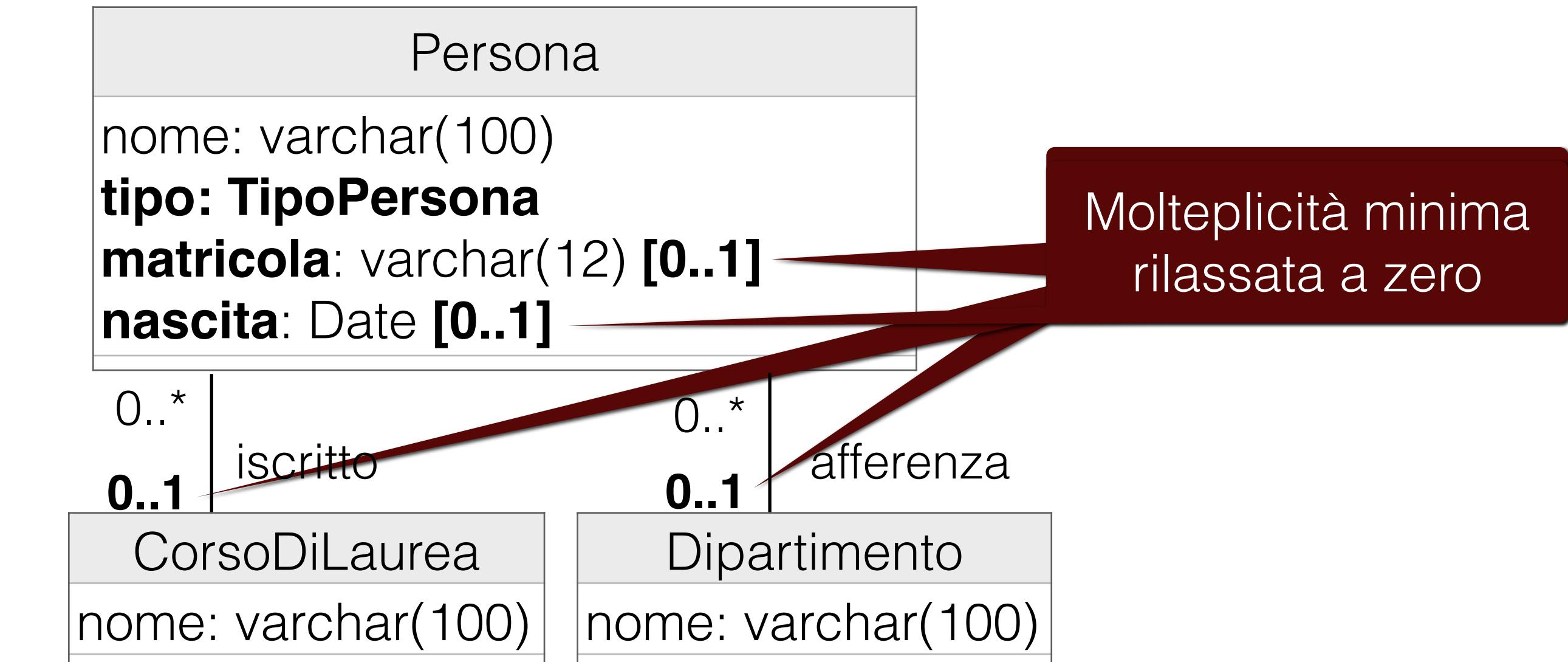
Vincoli esterni: per ogni istanza p di Persona:

1. $p.\text{tipo} = \text{'Studente'}$ se e solo se $p.\text{matricola}$ è valorizzato:
 $\forall p \text{ Persona}(p) \rightarrow [\text{tipo}(p, \text{'Studente'}) \leftrightarrow \exists m \text{ matricola}(p, m)]$
2. $p.\text{tipo} = \text{'Studente'}$ se e solo se p è coinvolto in un link “iscritto”:
 $\forall p \text{ Persona}(p) \rightarrow [\text{tipo}(p, \text{'Studente'}) \leftrightarrow \exists c \text{ iscritto}(c, p)]$

Ristrutturazione di generalizzazioni per fusione **in caso di sottoclassi disgiunte**



Più vincoli esterni da gestire



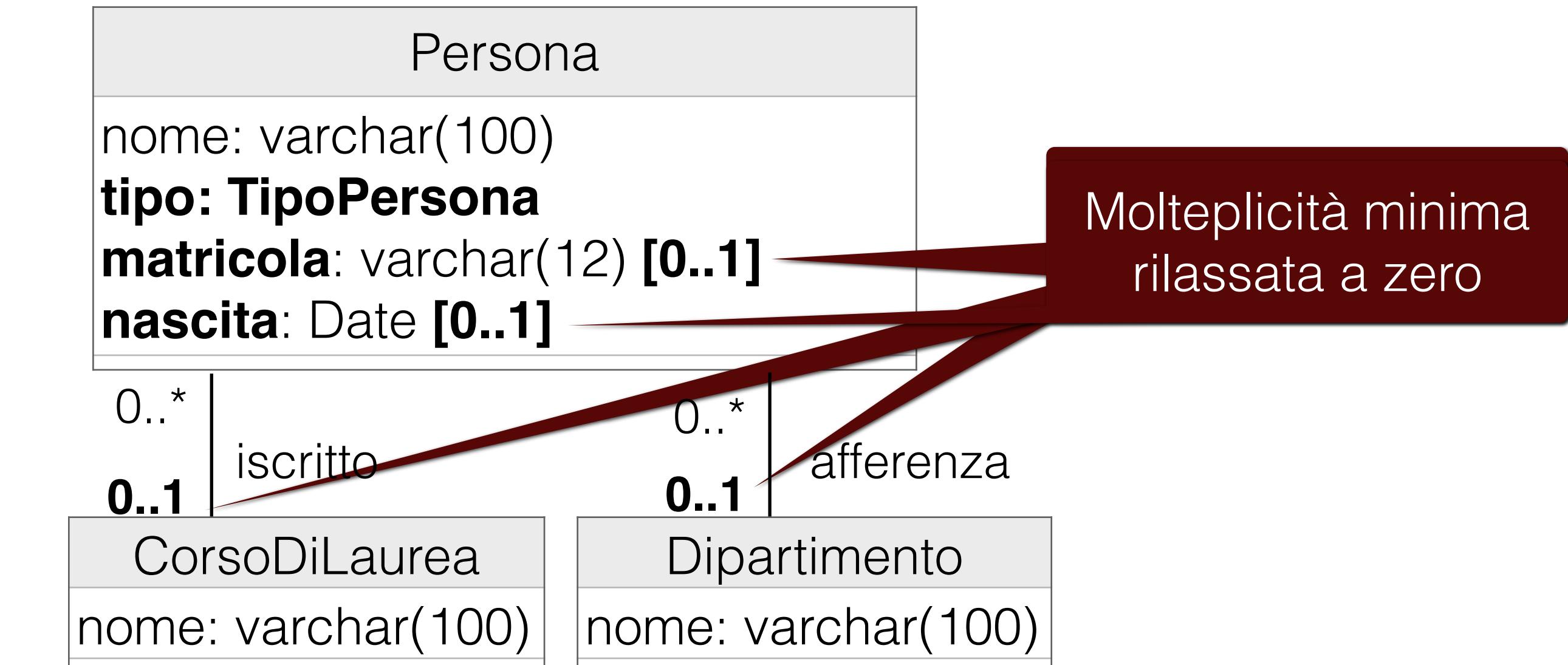
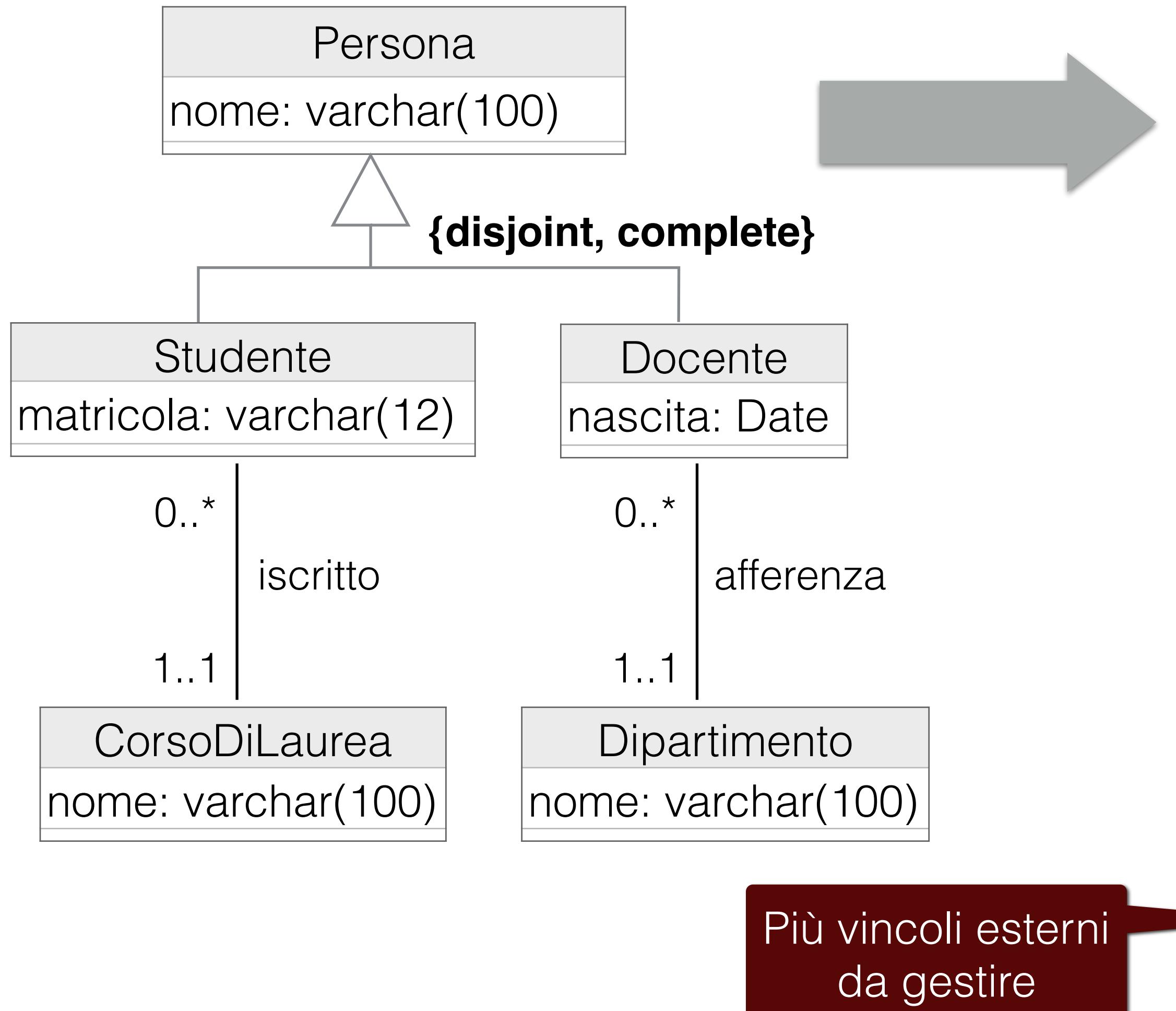
CREATE TYPE TipoPersona AS ENUM
('Persona' , 'Studente' , 'Docente');
non dobbiamo aggiunger un vincolo esterno per associazione nel caso di molteolicità 0..1?

Vincoli esterni: per ogni istanza p di Persona:

1. p.tipo = 'Studente' se e solo se p.matricola è valorizzato:
 $\forall p \text{ Persona}(p) \rightarrow [\text{tipo}(p, \text{'Studente'}) \leftrightarrow \exists m \text{ matricola}(p, m)]$
2. p.tipo = 'Studente' se e solo se p è coinvolto in un link "iscritto":
 $\forall p \text{ Persona}(p) \rightarrow [\text{tipo}(p, \text{'Studente'}) \leftrightarrow \exists c \text{ iscritto}(c, p)]$
3. p.tipo = 'Docente' se e solo se p.nascita è valorizzato:
 $\forall p \text{ Persona}(p) \rightarrow [\text{tipo}(p, \text{'Docente'}) \leftrightarrow \exists n \text{ nascita}(p, n)]$
4. p.tipo = 'Docente' se e solo se p è coinvolto in un link "afferenza":
 $\forall p \text{ Persona}(p) \rightarrow [\text{tipo}(p, \text{'Docente'}) \leftrightarrow \exists d \text{ afferenza}(d, p)]$

Molteplicità minima rilassata a zero

Ristrutturazione di generalizzazioni per fusione **in caso di sottoclassi disgiunte e complete**



```

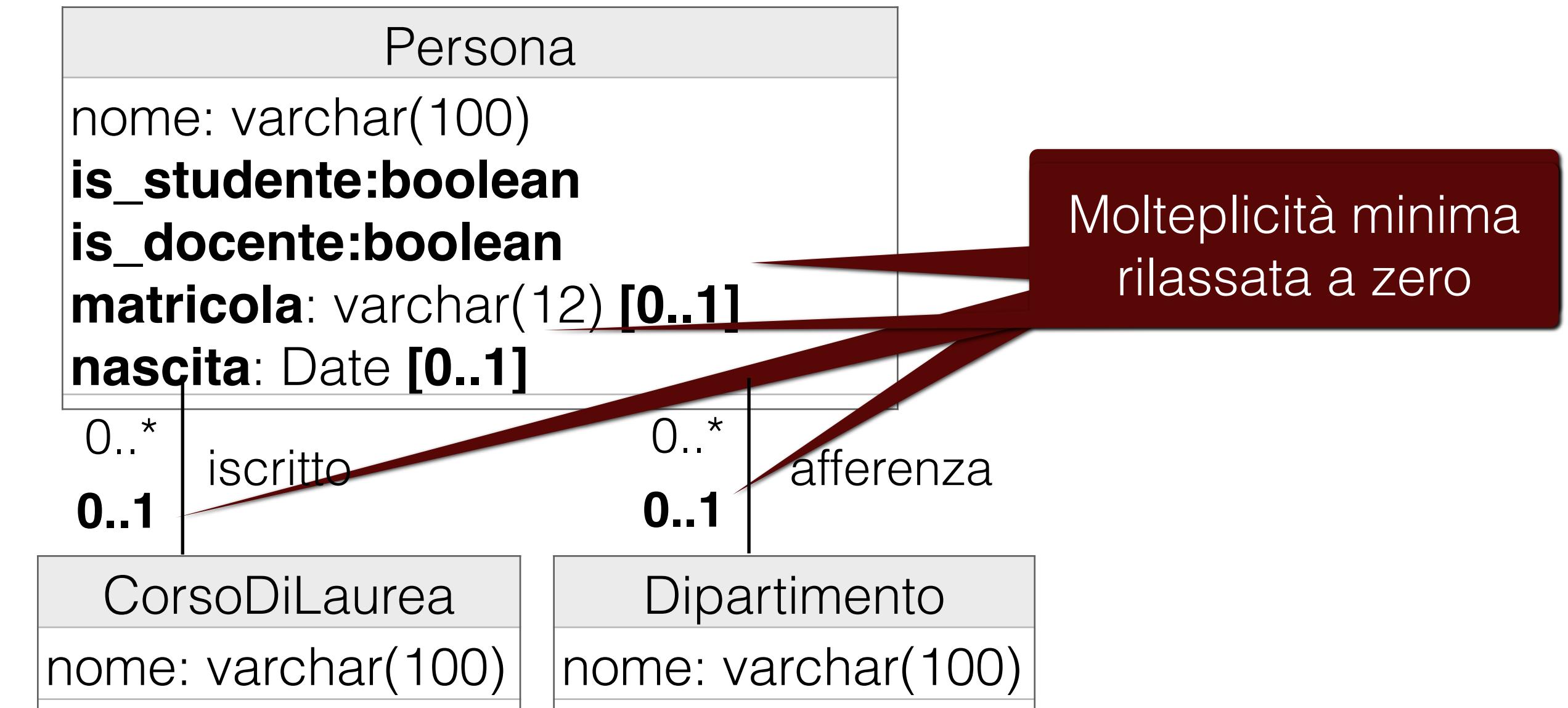
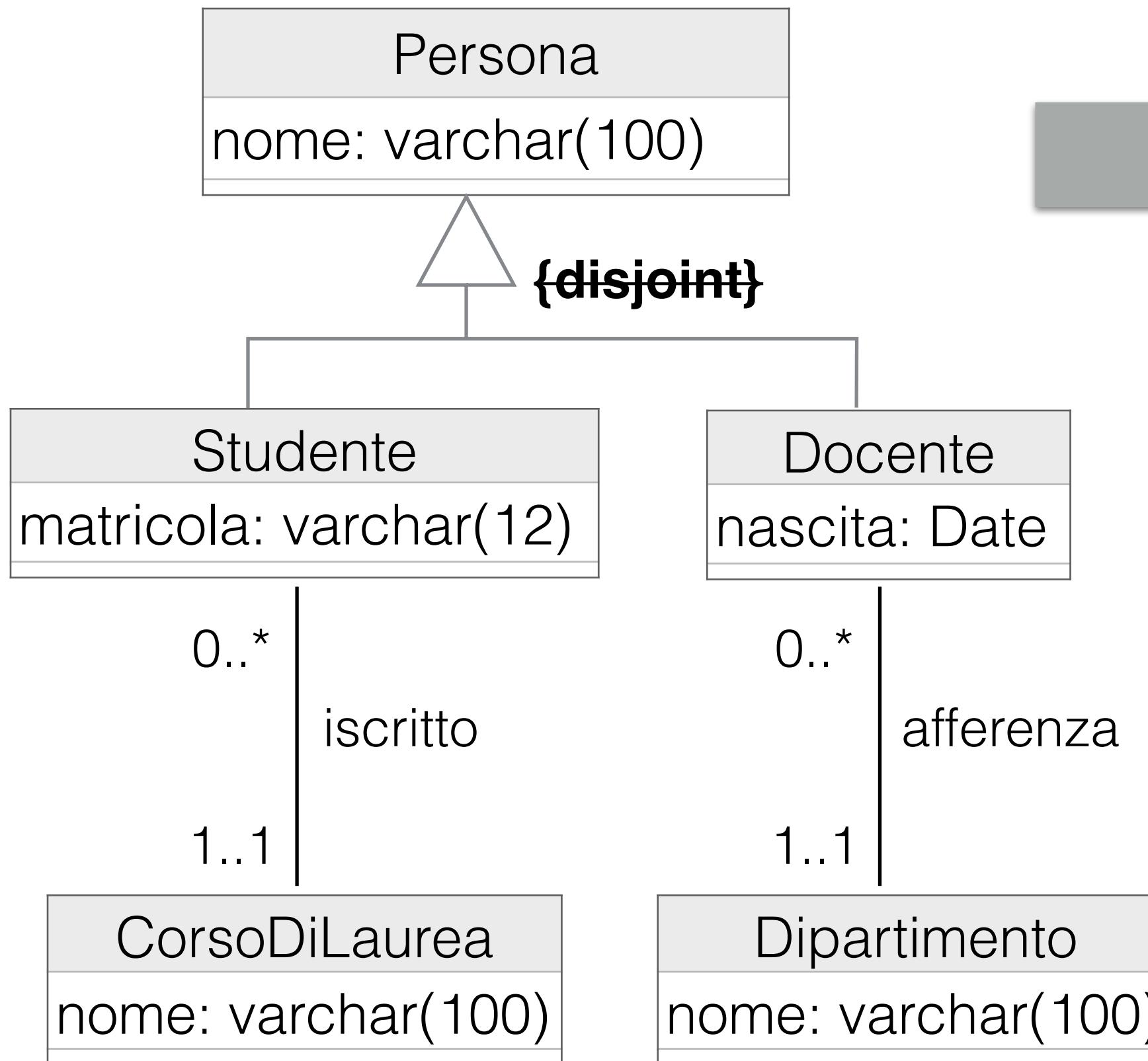
CREATE TYPE TipoPersona AS ENUM
  ('Persona', 'Studente', 'Docente');
  
```

Vincoli esterni: per ogni istanza p di Persona:

1. p.tipo = 'Studente' se e solo se p.matricola è valorizzato:
 $\forall p \text{ Persona}(p) \rightarrow [\text{tipo}(p, 'Studente') \leftrightarrow \exists m \text{ matricola}(p, m)]$
2. p.tipo = 'Studente' se e solo se p è coinvolto in un link "iscritto":
 $\forall p \text{ Persona}(p) \rightarrow [\text{tipo}(p, 'Studente') \leftrightarrow \exists c \text{ iscritto}(c, p)]$
3. p.tipo = 'Docente' se e solo se p.nascita è valorizzato:
 $\forall p \text{ Persona}(p) \rightarrow [\text{tipo}(p, 'Docente') \leftrightarrow \exists n \text{ nascita}(p, n)]$
4. p.tipo = 'Docente' se e solo se p è coinvolto in un link "afferenza":
 $\forall p \text{ Persona}(p) \rightarrow [\text{tipo}(p, 'Docente') \leftrightarrow \exists d \text{ afferenza}(d, p)]$

Più vincoli esterni da gestire

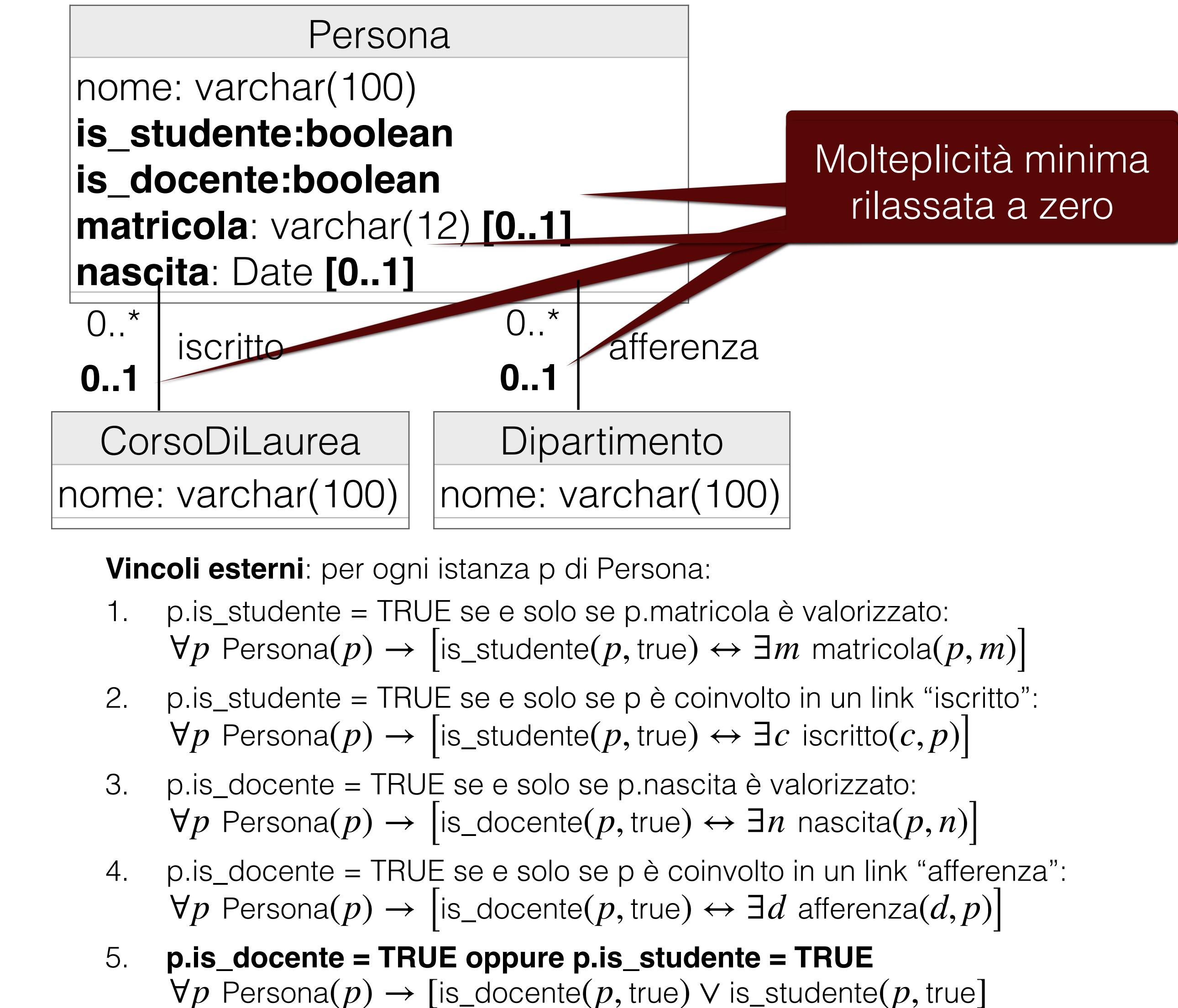
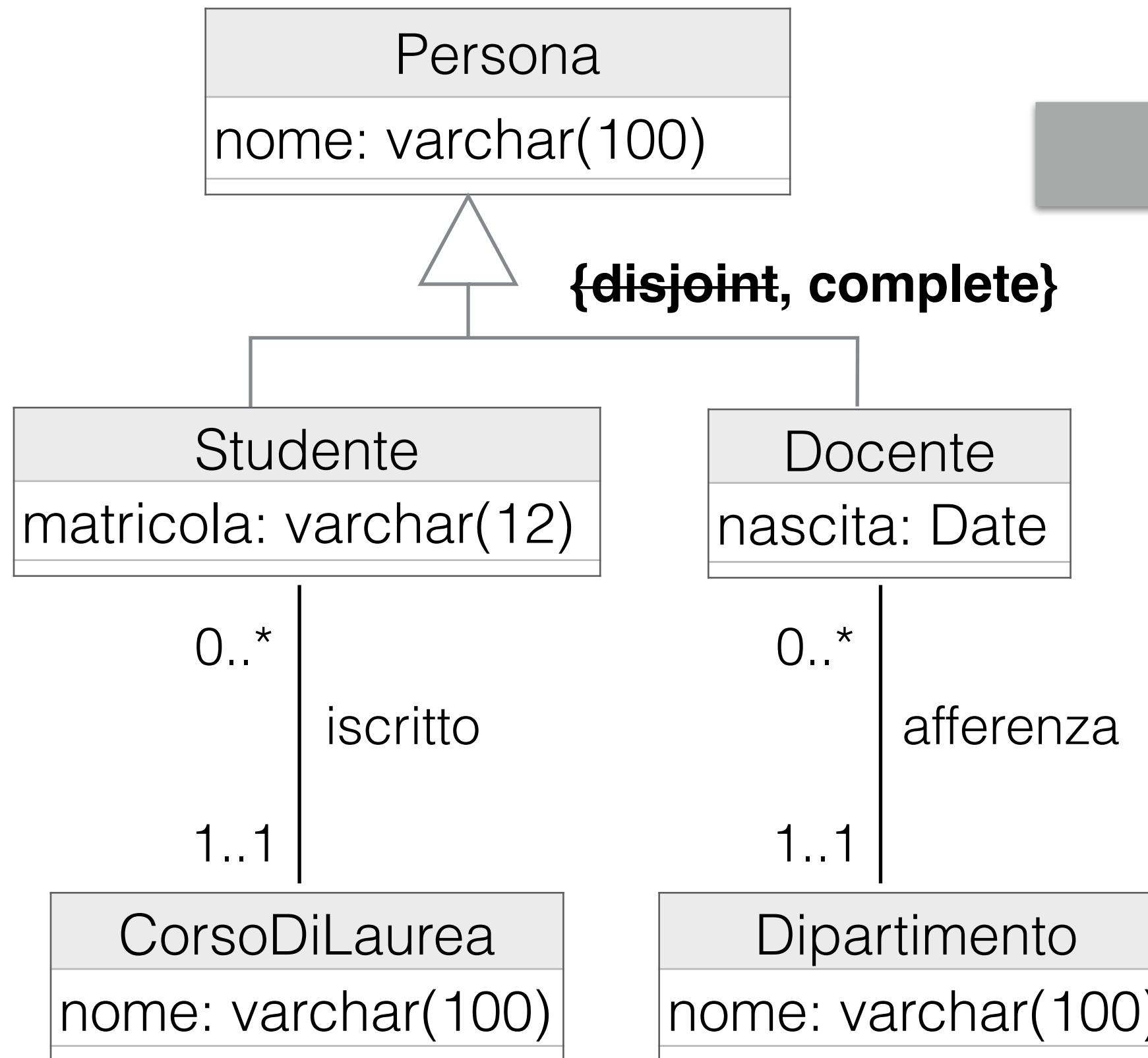
Ristrutturazione di generalizzazioni per fusione **in caso di sottoclassi disgiunte**



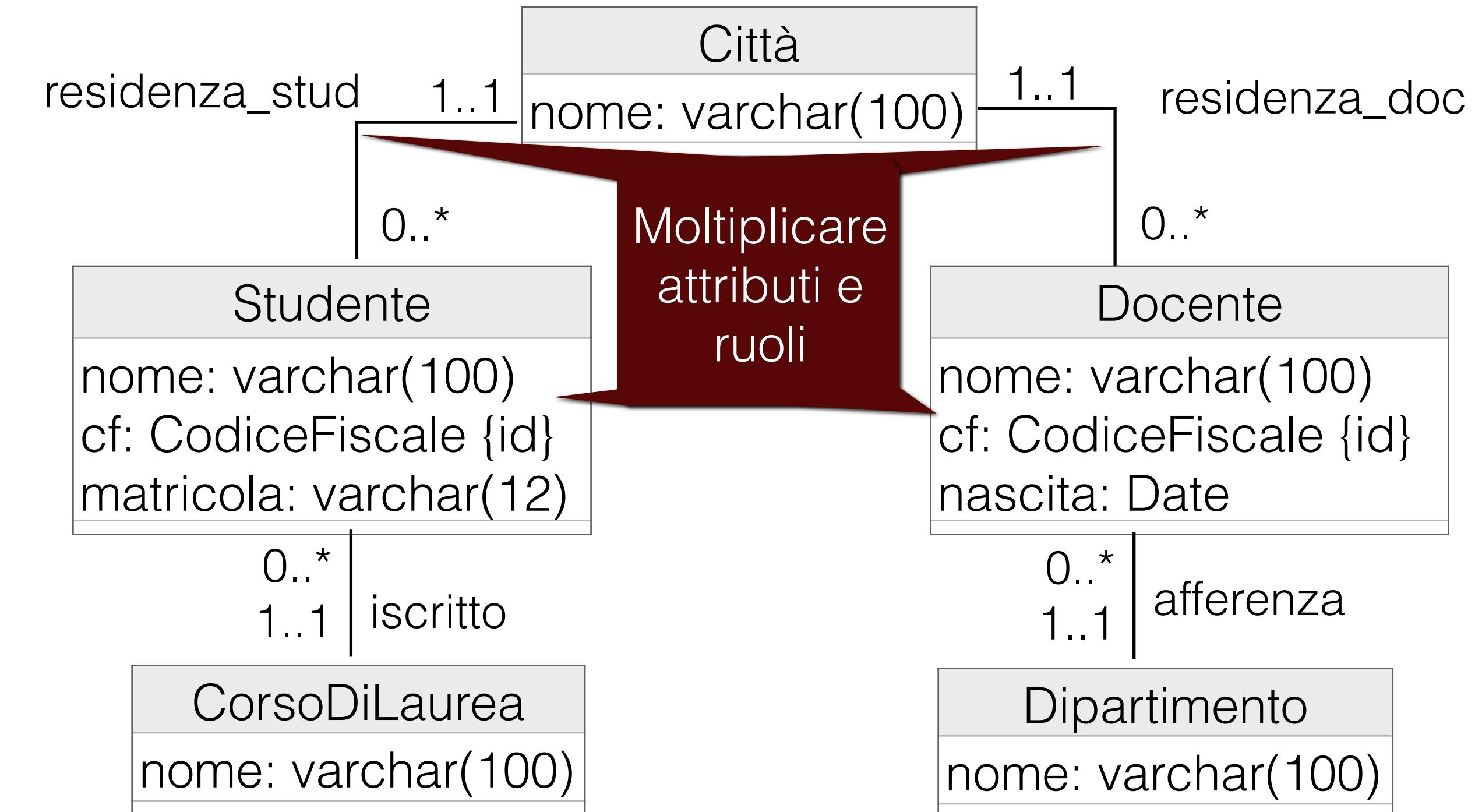
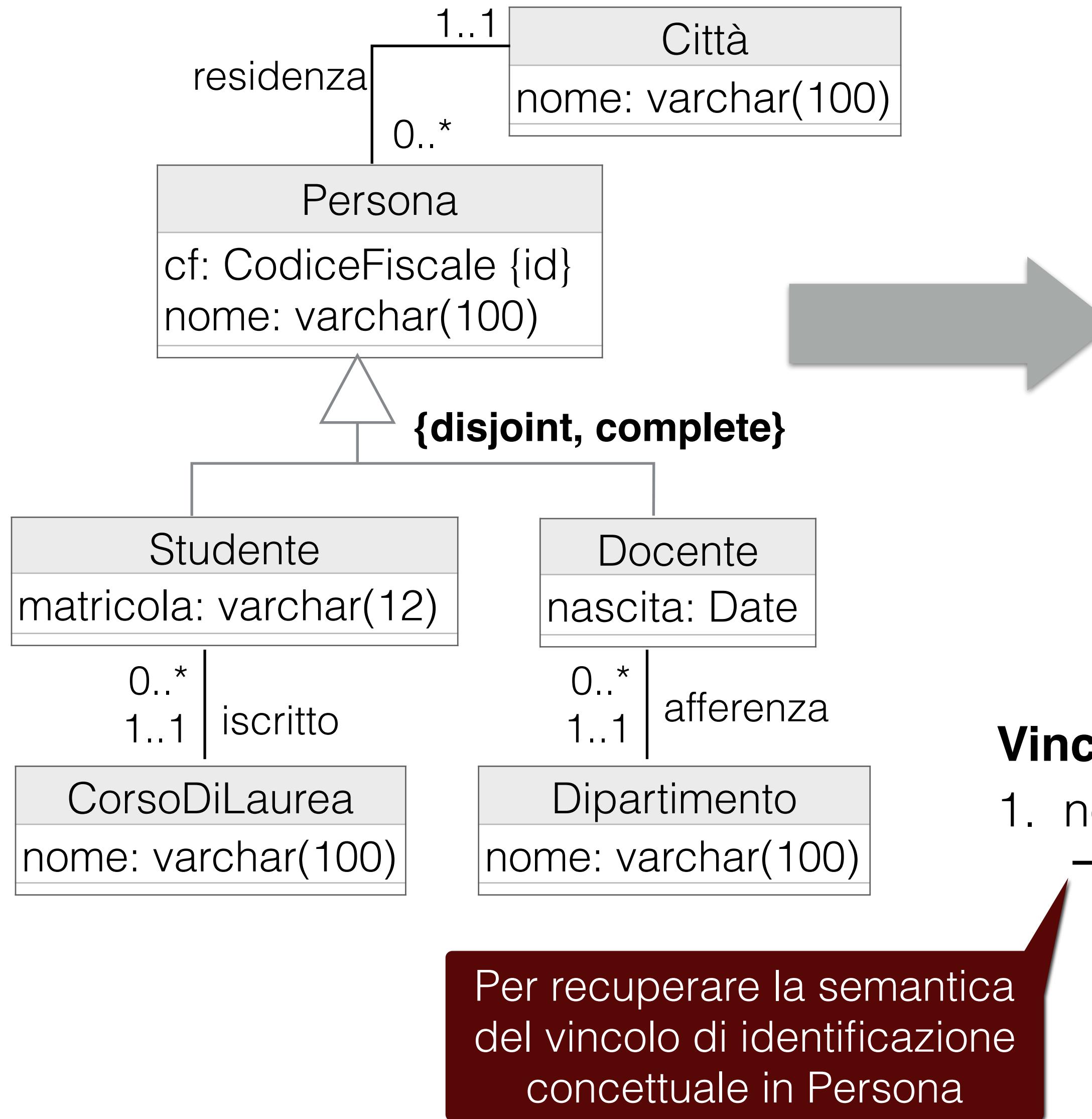
Vincoli esterni: per ogni istanza p di Persona:

1. $p.\text{is_studente} = \text{TRUE}$ se e solo se $p.\text{matricola}$ è valorizzato:
 $\forall p \text{ Persona}(p) \rightarrow [\text{is_studente}(p, \text{true}) \leftrightarrow \exists m \text{ matricola}(p, m)]$
2. $p.\text{is_studente} = \text{TRUE}$ se e solo se p è coinvolto in un link “iscritto”:
 $\forall p \text{ Persona}(p) \rightarrow [\text{is_studente}(p, \text{true}) \leftrightarrow \exists c \text{ iscritto}(c, p)]$
3. $p.\text{is_docente} = \text{TRUE}$ se e solo se $p.\text{nascita}$ è valorizzato:
 $\forall p \text{ Persona}(p) \rightarrow [\text{is_docente}(p, \text{true}) \leftrightarrow \exists n \text{ nascita}(p, n)]$
4. $p.\text{is_docente} = \text{TRUE}$ se e solo se p è coinvolto in un link “afferenza”:
 $\forall p \text{ Persona}(p) \rightarrow [\text{is_docente}(p, \text{true}) \leftrightarrow \exists d \text{ afferenza}(d, p)]$

Ristrutturazione di generalizzazioni per fusione **in caso di sottoclassi disgiunte**



Ristrutturazione di generalizzazioni per divisione **in caso di sottoclassi disgiunte e complete**



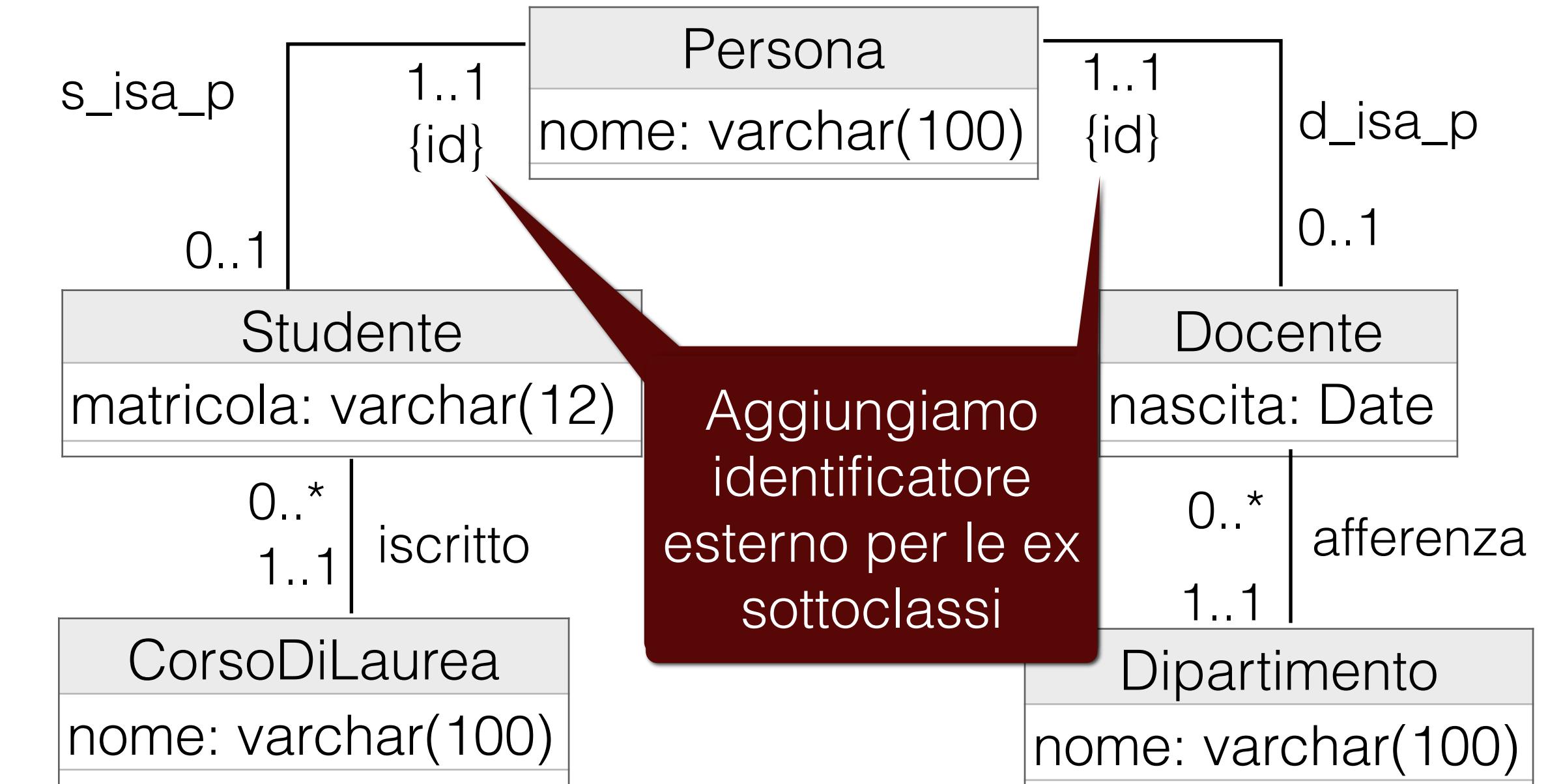
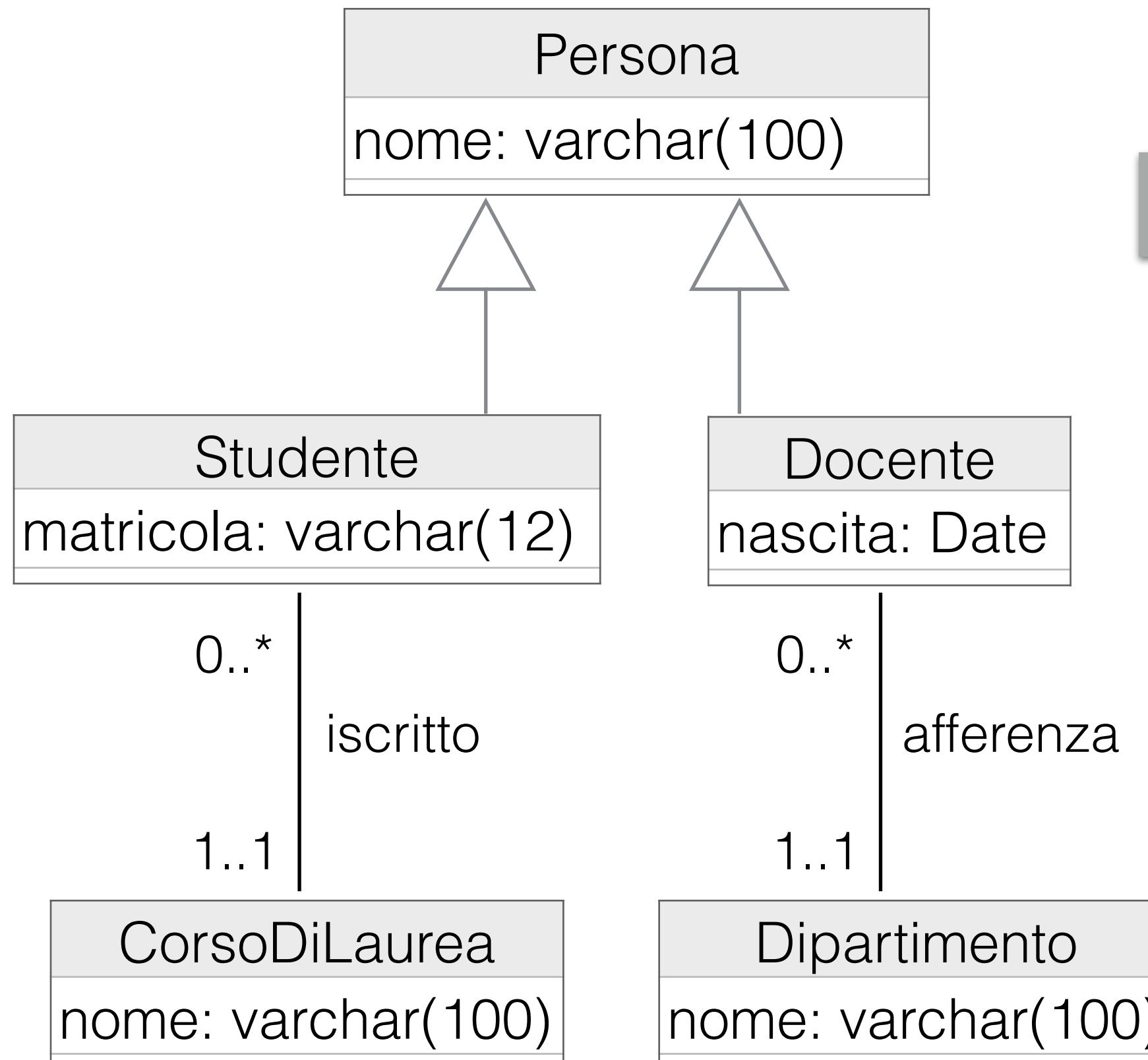
Vincoli esterni:

1. non devono esistere $s:\text{Studente}$ e $d:\text{Docente}$ tali che $s.cf = d.cf$:
 $\neg \exists s, d, v \text{ Studente}(s) \wedge \text{Docente}(d) \wedge cf(s, v) \wedge cf(d, v)$

Approccio problematico per generalizzazioni non complete e/o non disgiunte

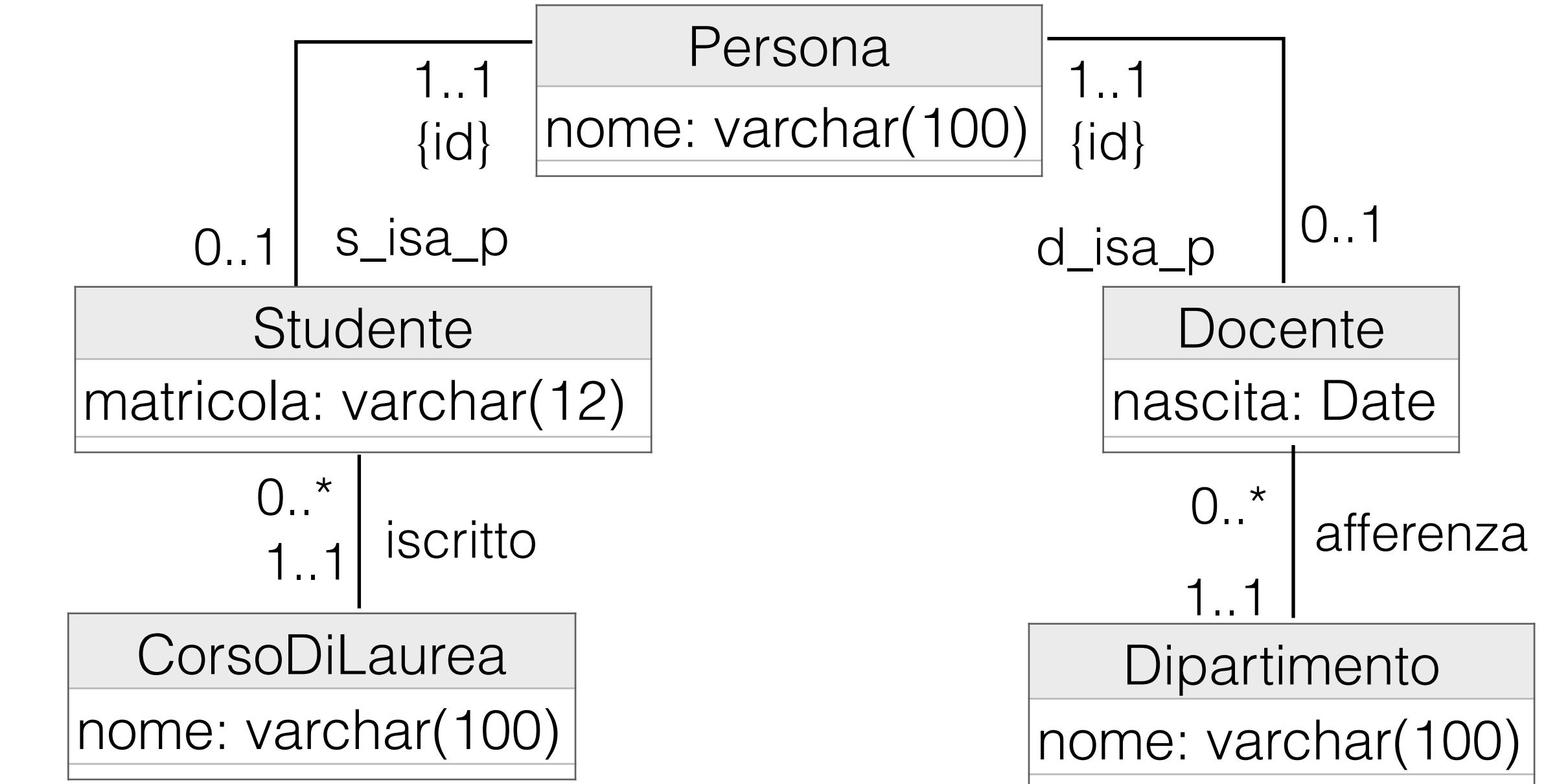
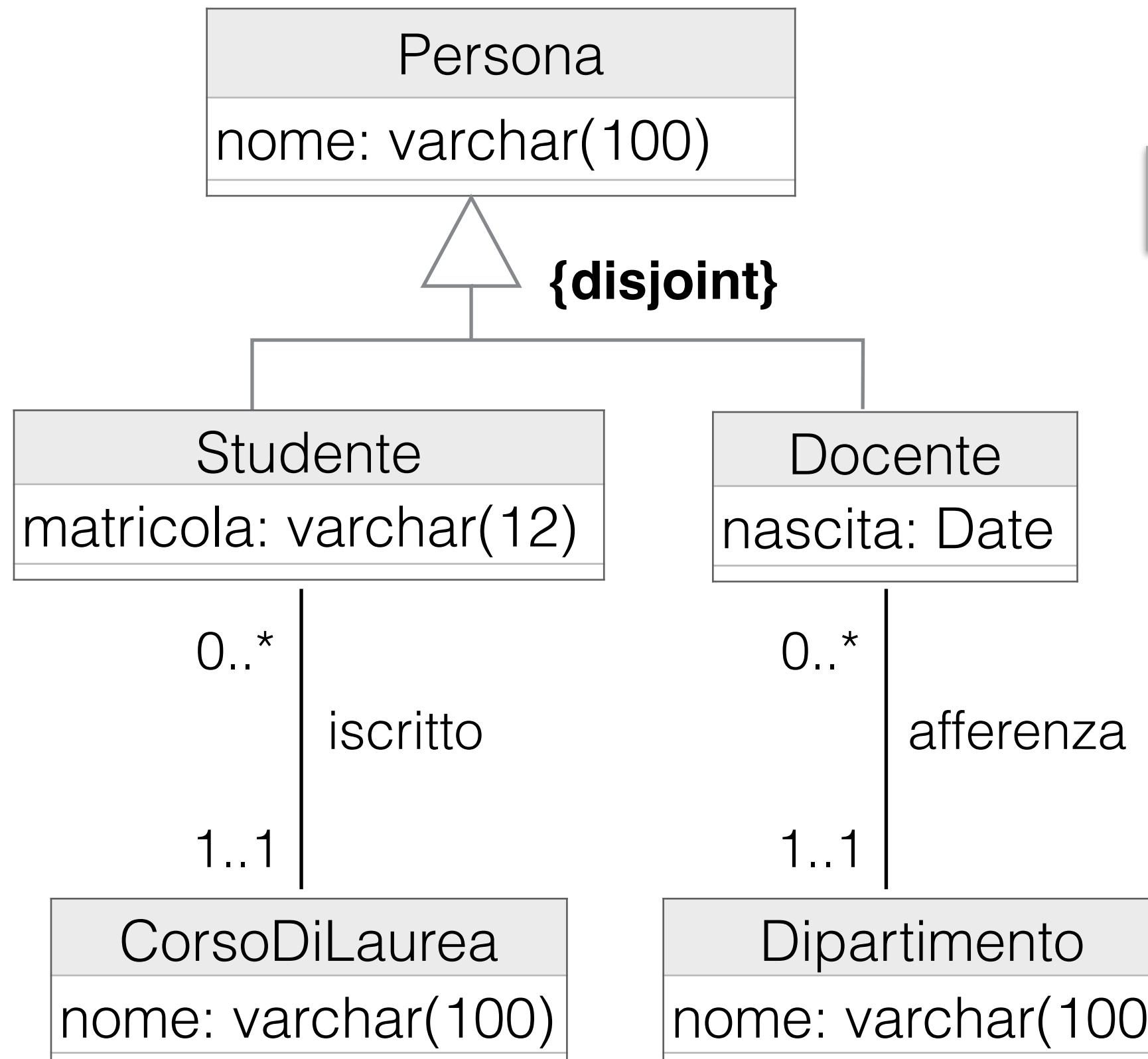
In generale: approccio da usare con cautela!

Ristrutturazione di relazioni is-a indipendenti o generalizzazioni non disgiunte e non complete



Struttura perfettamente mantenuta

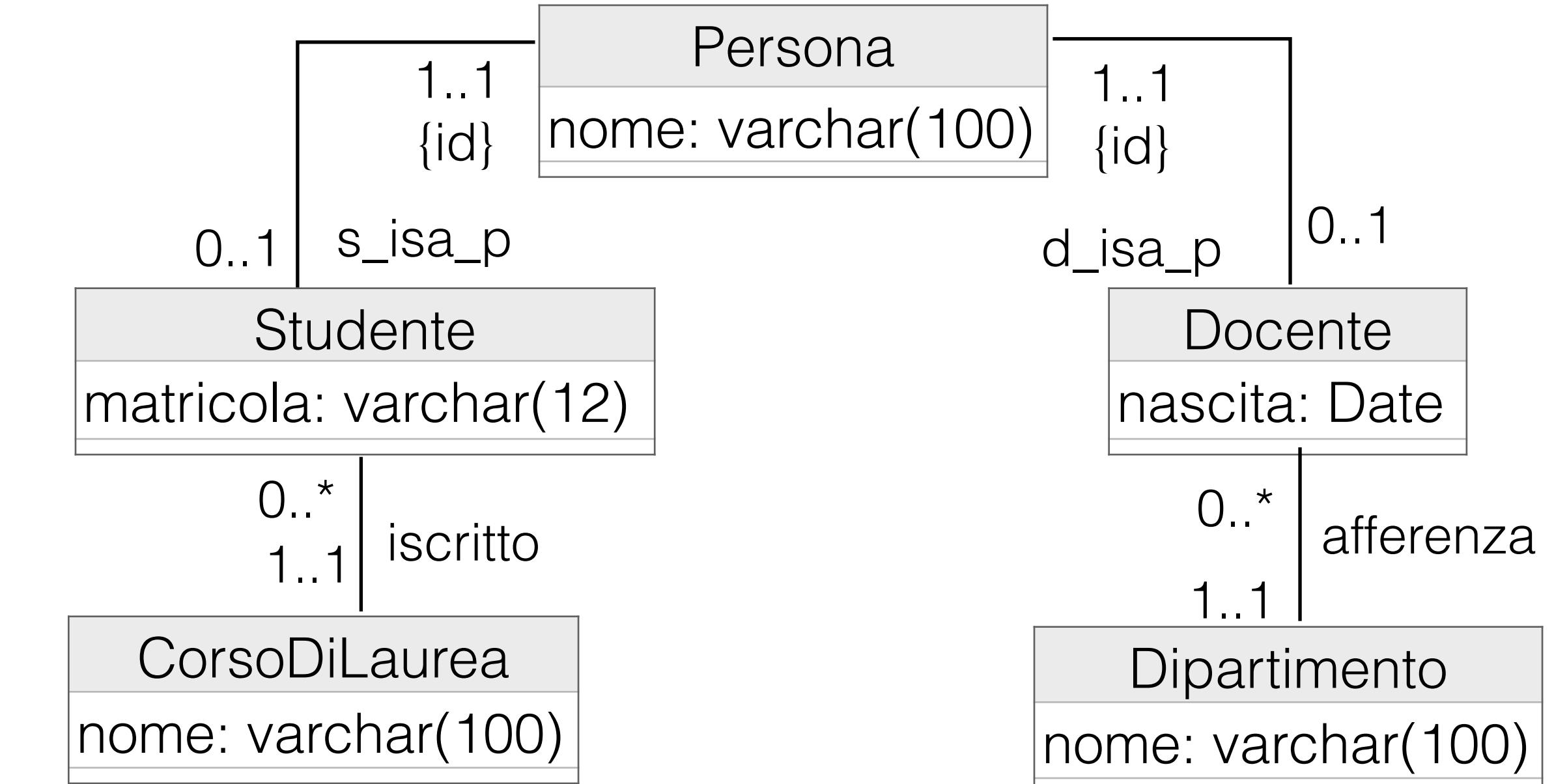
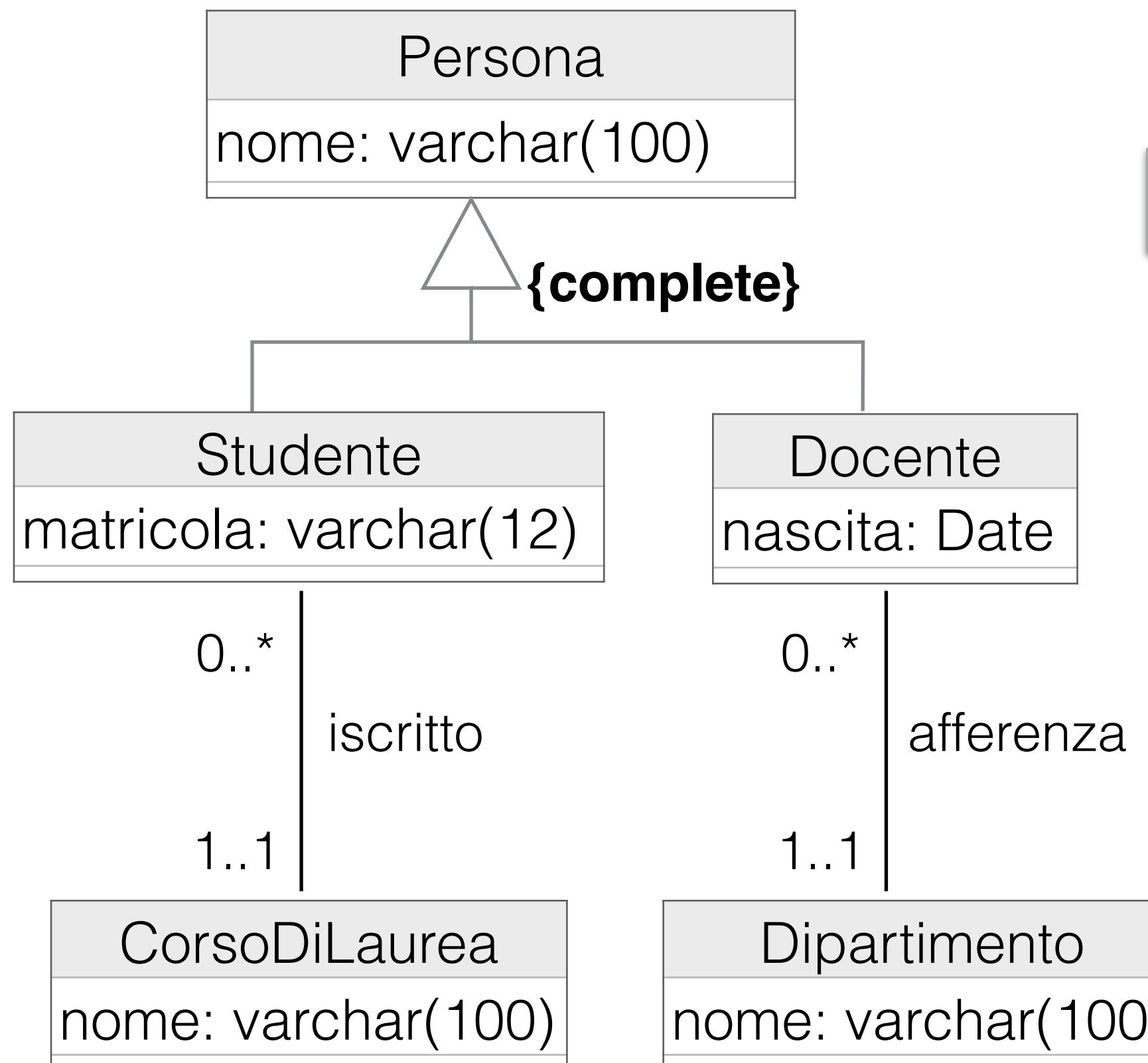
Ristrutturazione **in caso di sottoclassi disgiunte**



Vincoli esterni:

1. Disgiunzione: non devono esistere $p: \text{Persona}$, $s: \text{Studente}$ e $d: \text{Docente}$ tali che $(s, p): s_{\text{isa}}_p$ e $(d, p): d_{\text{isa}}_p$:
 $\neg \exists p, s, d \text{ Persona}(p) \wedge \text{Studente}(s) \wedge \text{Docente}(d) \wedge s_{\text{isa}}_p(s, p) \wedge d_{\text{isa}}_p(d, p)$
ma non viene implicata dalla tipizzazione degli attributi?

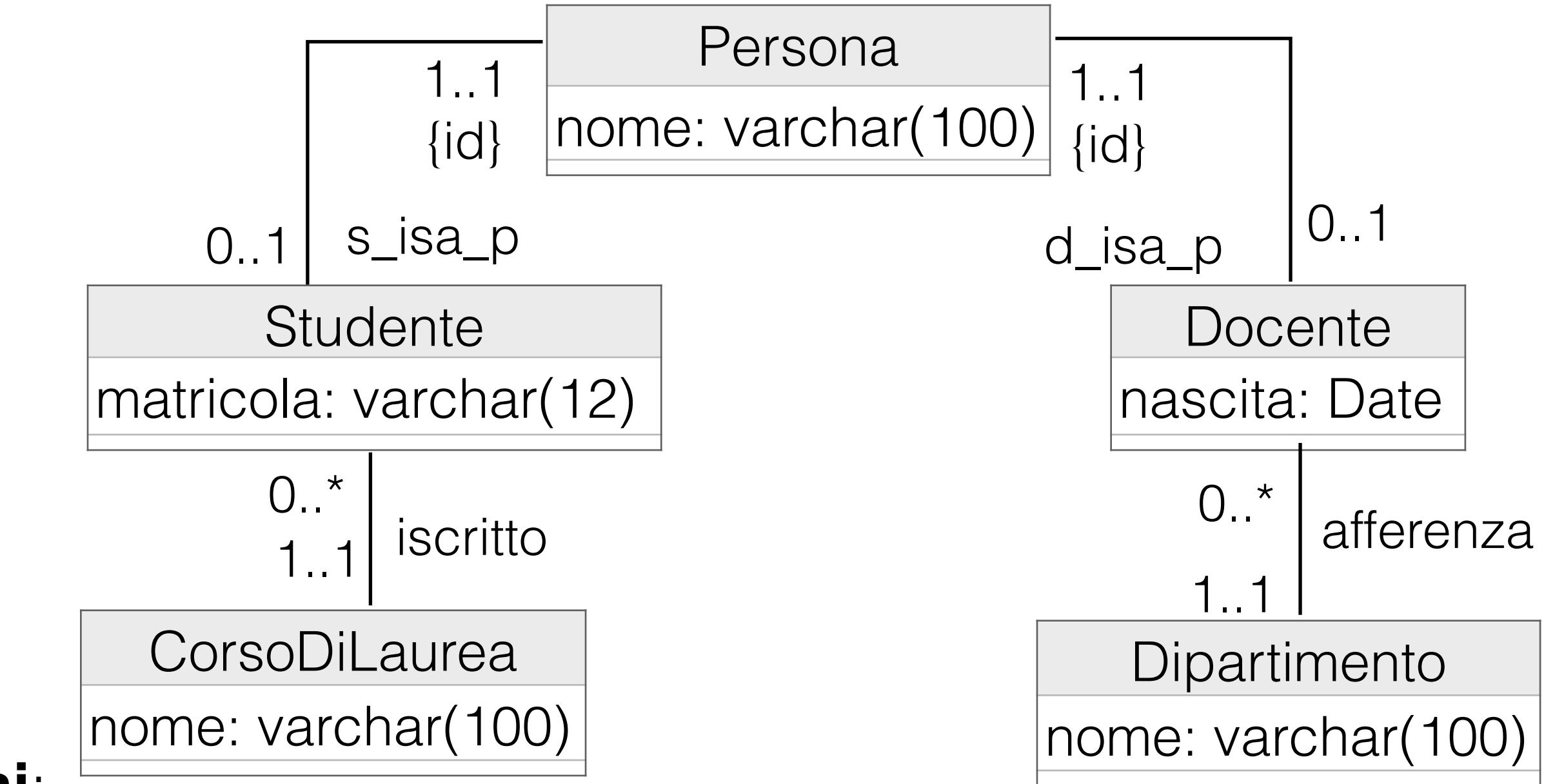
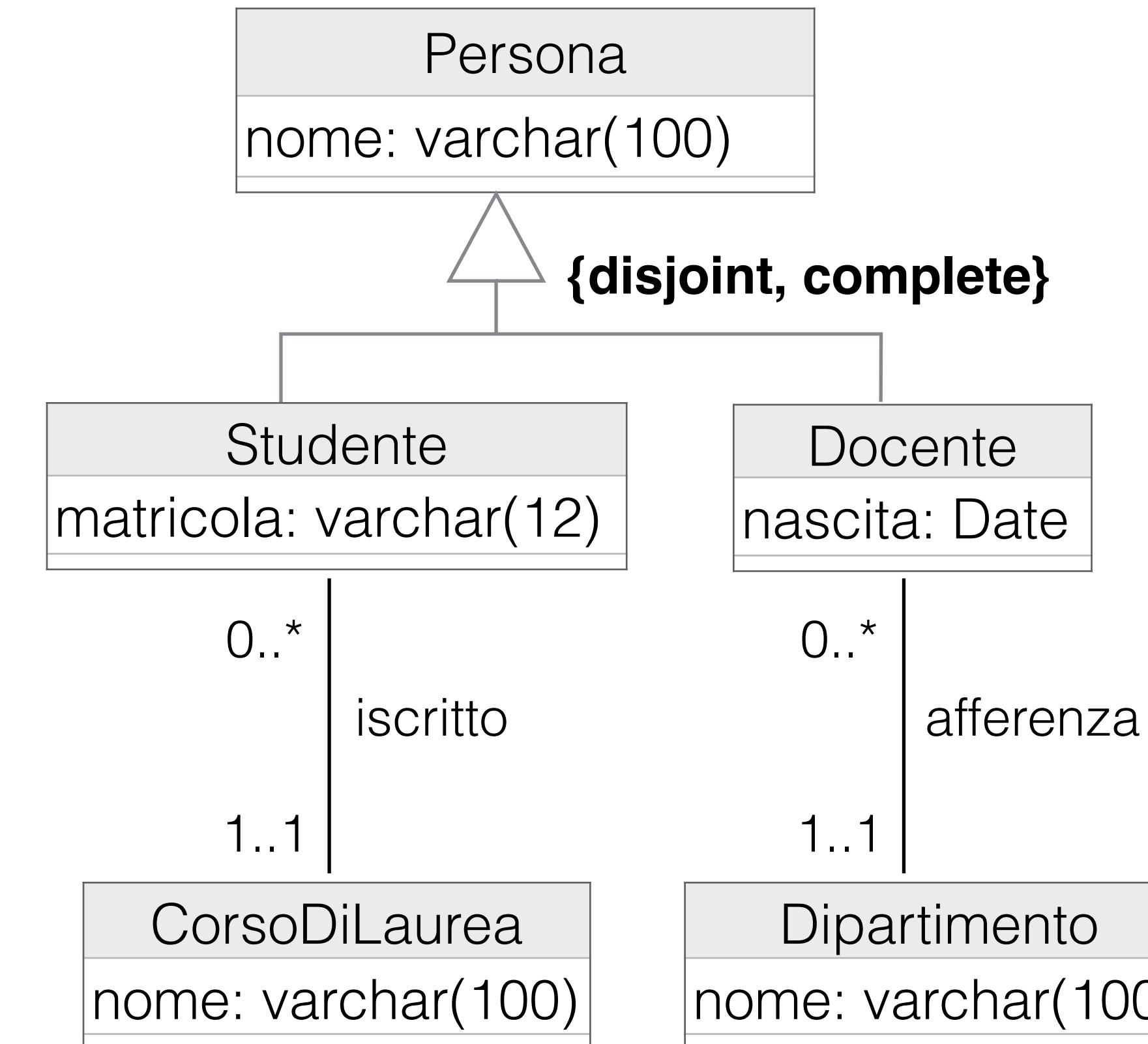
Ristrutturazione **in caso di sottoclassi complete**



Vincoli esterni:

1. Completezza: per ogni p :**Persona**:
 - esiste s :**Studente** per cui: (s, p) : **s_isa_p**
 - oppure esiste d :**Docente** per cui: (d, p) : **d_isa_p**:
$$\forall p \text{ Persona}(p) \rightarrow [\exists s \text{ s_isa_p}(s, p)] \vee [\exists d \text{ d_isa_p}(d, p)]$$

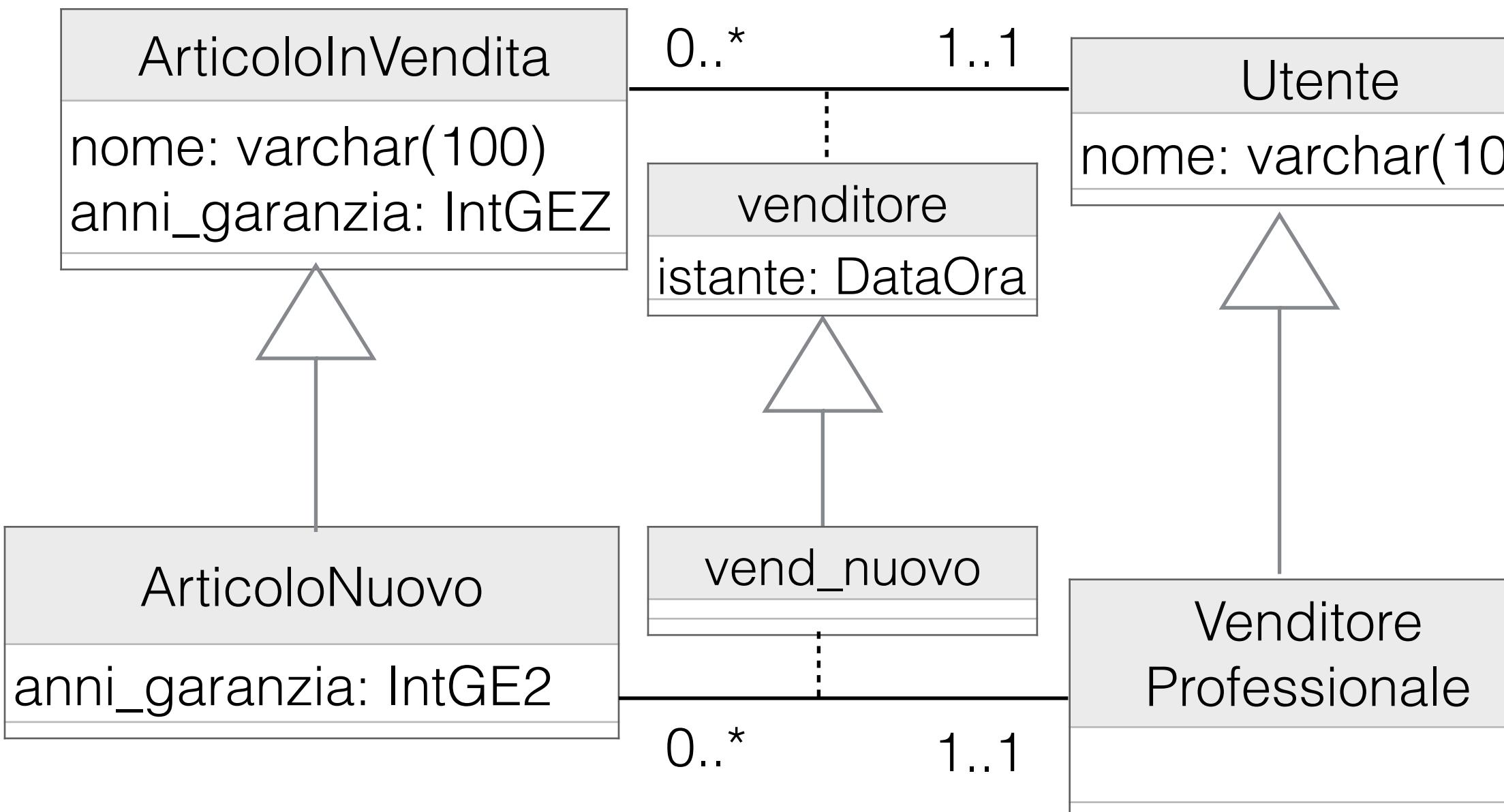
Ristrutturazione **in caso di sottoclassi disgiunte e complete**



Vincoli esterni:

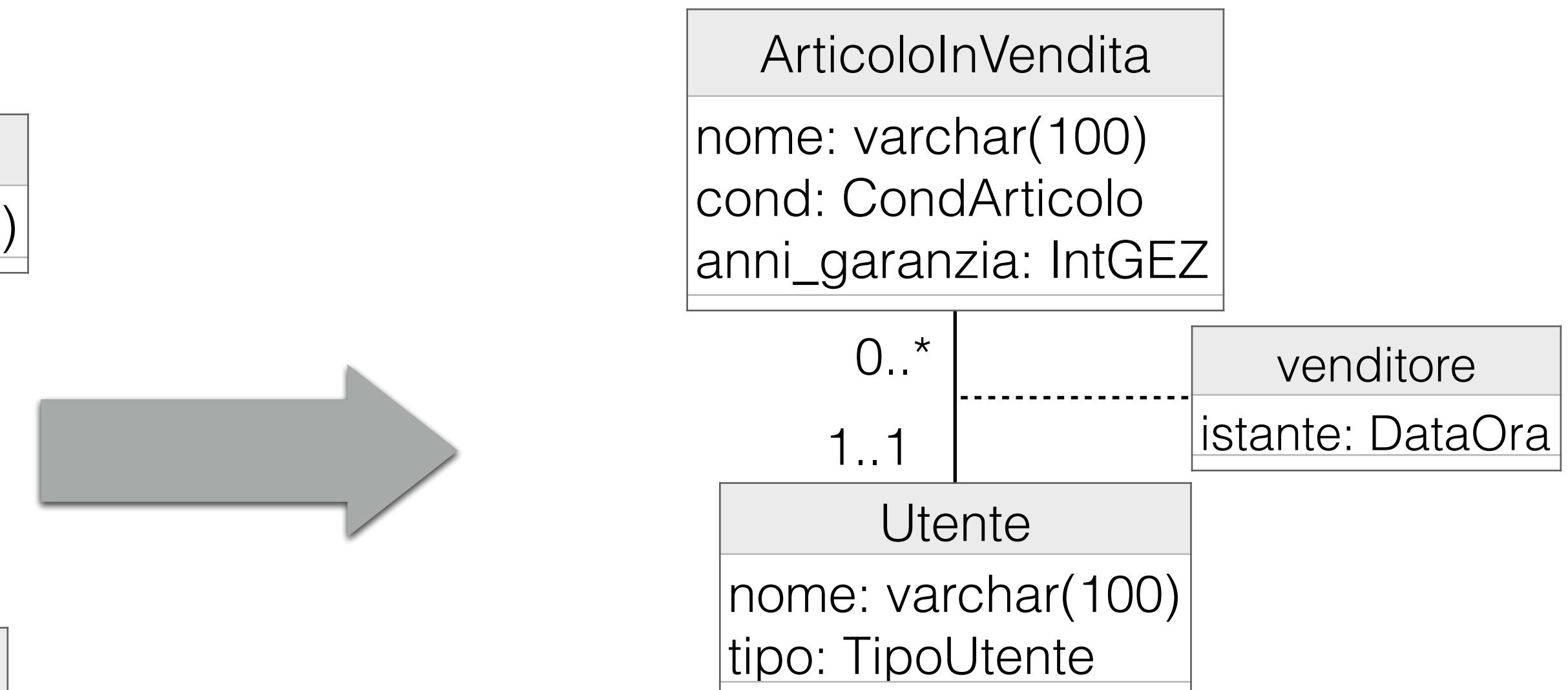
1. Disgiunzione: non devono esistere p :Persona, s :Studente e d :Docente tali che $(s, p) : s_isa_p$ e $(d, p) : d_isa_p$:
 $\neg \exists p, s, d \text{ Persona}(p) \wedge \text{Studente}(s) \wedge \text{Docente}(d) \wedge s_isa_p(s, p) \wedge d_isa_p(d, p)$
2. Completezza: per ogni p :Persona:
 - esiste s :Studente per cui: $(s, p) : s_isa_p$
 - oppure esiste d :Docente per cui: $(d, p) : d_isa_p$ $\forall p \text{ Persona}(p) \rightarrow [\exists s \text{ s_isa_p}(s, p)] \vee [\exists d \text{ d_isa_p}(d, p)]$

Ristrutturazione **in caso le relazioni is-a tra classi siano state ristrutturate per fusione**



```

CREATE DOMAIN IntGEZ AS Integer
  (check value >= 0);
CREATE DOMAIN IntGE2 AS Integer
  (check value >= 2);
  
```



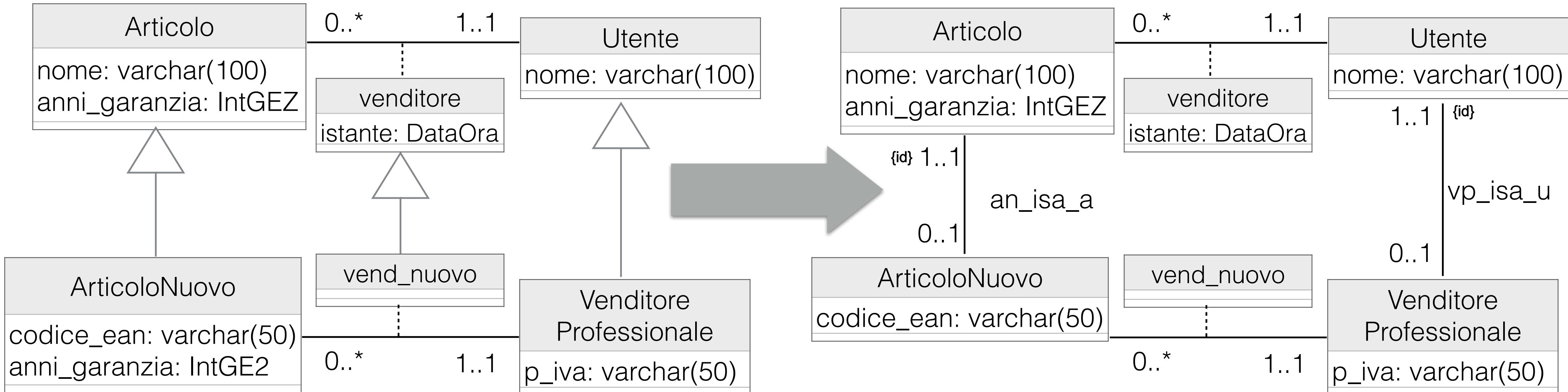
```

CREATE TYPE CondArticolo AS
  ENUM ('nuovo', 'usato');
CREATE TYPE TipoUtente AS
  ENUM ('privato', 'prof');
  
```

Vincoli esterni:

- per ogni a :ArticoloInVendita:
 - se $a.cond = 'nuovo'$ allora $a.anni_garanzia \geq 2$:
 $\forall a, g \text{ ArticoloInVendita}(a) \wedge anni_garanzia(a, g) \wedge cond(a, nuovo) \rightarrow g \geq 2$
 - se $a.cond = 'nuovo'$ allora il link (a,u) :venditore nel quale ' a ' è coinvolto è tale che $u.tipo = 'prof'$
 $\forall a, u \text{ ArticoloInVendita}(a) \wedge venditore(a, u) \wedge cond(a, nuovo) \rightarrow tipo(u, prof)$

Ristrutturazione **in caso le relazioni is-a tra classi siano state ristrutturate per sostituzione con associazioni**



Vincoli esterni:

- per ogni $an:ArticoloNuovo$, l'istanza $a:Articolo$ tale che $(an, a):an_isa_a$ deve avere: $a.anni_garanzia \geq 2$
 $\forall an, a, g \text{ ArticoloNuovo}(an) \wedge Articolo(a) \wedge an_isa_a(an, a) \wedge anni_garanzia(a, g) \rightarrow g \geq 2$
- per ogni $an:ArticoloNuovo$, $a:Articolo$, $vp:VendProf$, $u:Utente$ tali che:
 - $(an, vp): vend_nuovo$, $(an, a): an_isa_a$, $(vp, u): vp_isa_u$
 deve essere: $(a, u): venditore$:
 $\forall an, a, vp, u \text{ ArticoloNuovo}(an) \wedge Articolo(a) \wedge VendProf(vp) \wedge Utente(u) \wedge vend_nuovo(an, vp) \wedge an_isa_a(an, a) \wedge vp_isa_u(vp, u) \rightarrow venditore(a, u)$

```

CREATE DOMAIN IntGEZ AS Integer
  (check value >= 0);
CREATE DOMAIN IntGE2 AS Integer
  (check value >= 2);
  
```

Al termine del passo di ristrutturazione delle generalizzazioni, tutte le classi e le associazioni del diagramma ristrutturato definiscono insiemi di istanze (oggetti o link) disgiunte a coppie

Basi di Dati, Modulo 2

Laurea in Informatica

Parte: Cenni di ingegneria del sw
UNITÀ: B.3
Prof. Toni Mancini
Dipartimento di Informatica



SAPIENZA
UNIVERSITÀ DI ROMA

S.B.3.1

Basi di dati relazionali
Progettazione di basi dati relazionali
Produzione dello schema rel. con vincoli

Ristrutturazione del diagramma UML delle
classi

Identificatori di classe

Identifieri di classe

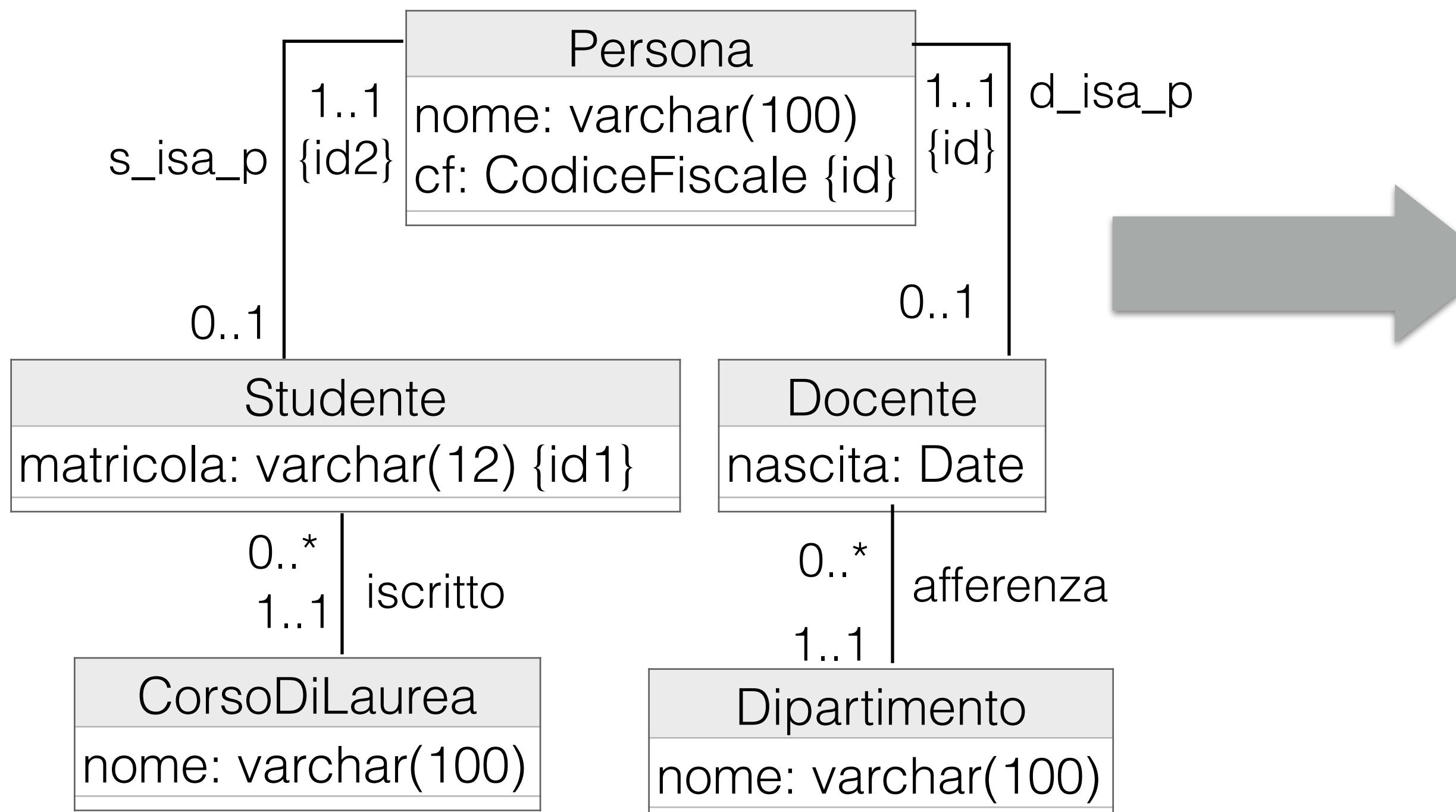
Obiettivo:

- Ristrutturare il diagramma UML concettuale delle classi in uno equivalente che **abbia almeno un identificatore per ogni classe**, che fungerà da chiave della futura tabella del DB, ed eleggere **un identificatore primario per ogni classe**, che fungherà da chiave primaria della futura tabella.

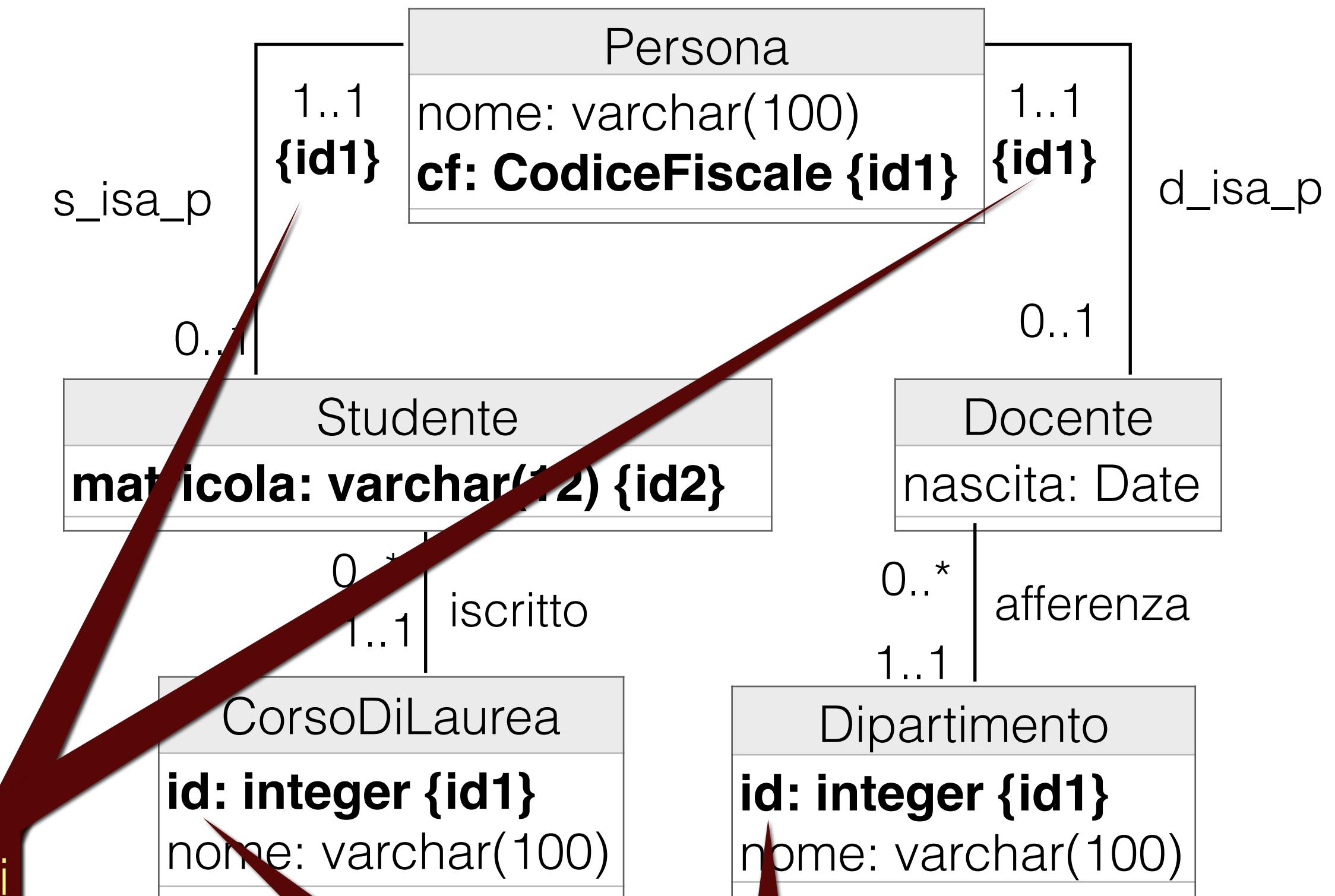
Metodologia:

- **Per ogni classe con identificatori concettuali**, decidere se uno tra quelli disponibili è di qualità sufficiente per formare una chiave primaria: **non deve essere troppo complesso**. Eleggerlo a identificatore primario.
- **Per ogni classe senza identificatori concettuali**, o per la quale non è stato scelto un identificatore primario al passo precedente, definire un **identificatore artificiale**.

Identificatori di classe (cont.)



Per le classi che erano sotto-classi di generalizzazioni ristrutturate mediante sostituzione con associazioni, questo identificatore dove essere il primario

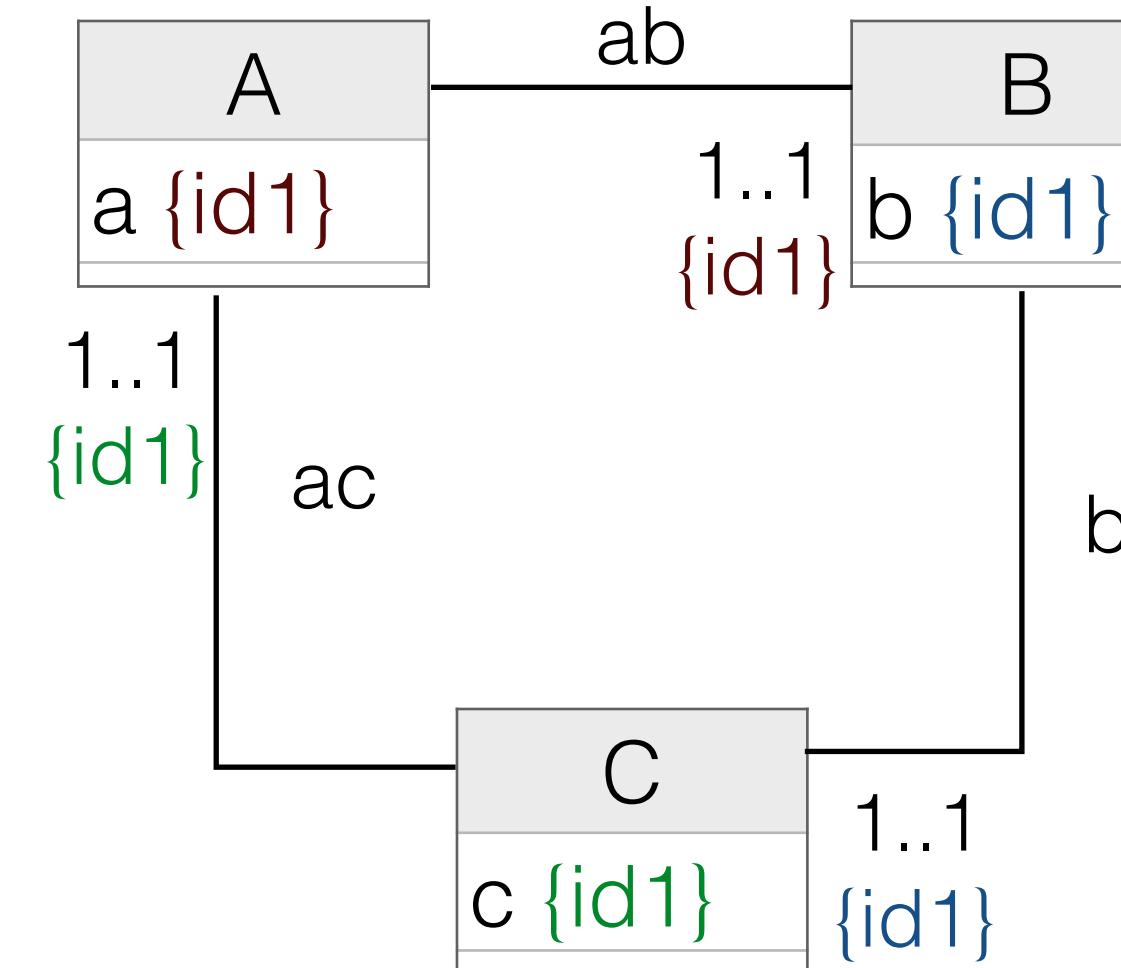


Identificatori artificiali, di nessuna rilevanza concettuale

Attributi e ruoli identificatori primari di una classe denotati con {id1}
Eventuali altri identificatori con {id2}, {id3}, etc.

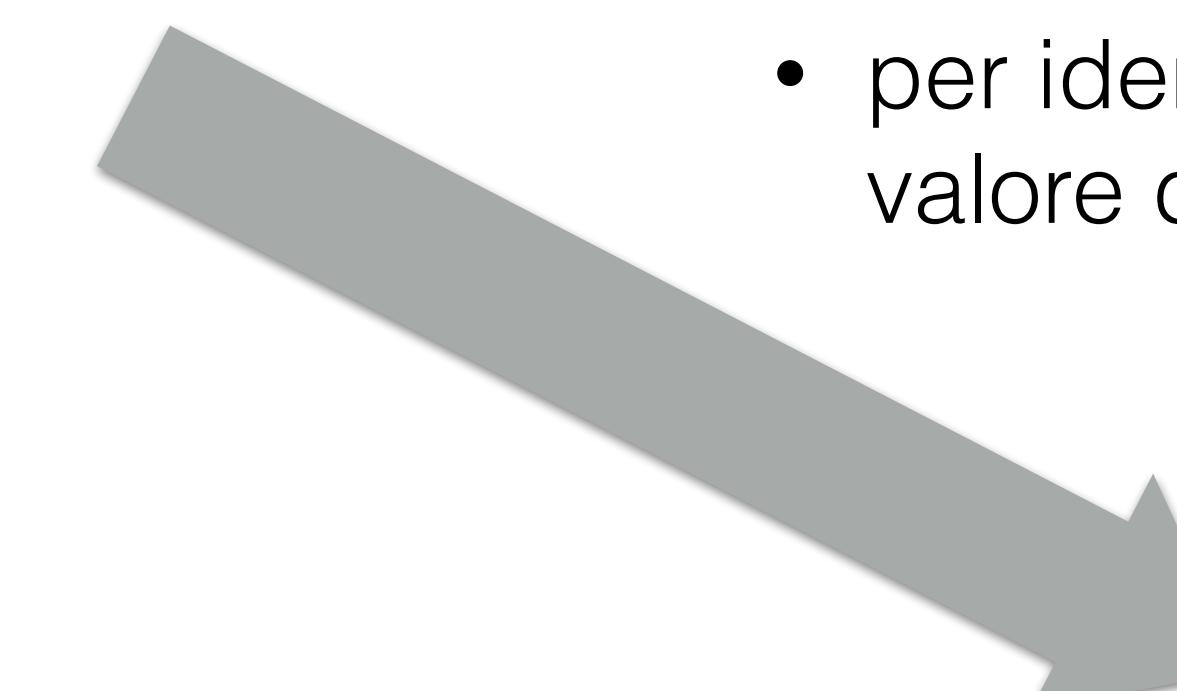
Identificatori di classe (cont.)

- Controllare che non ci siano cicli di identificatori primari esterni



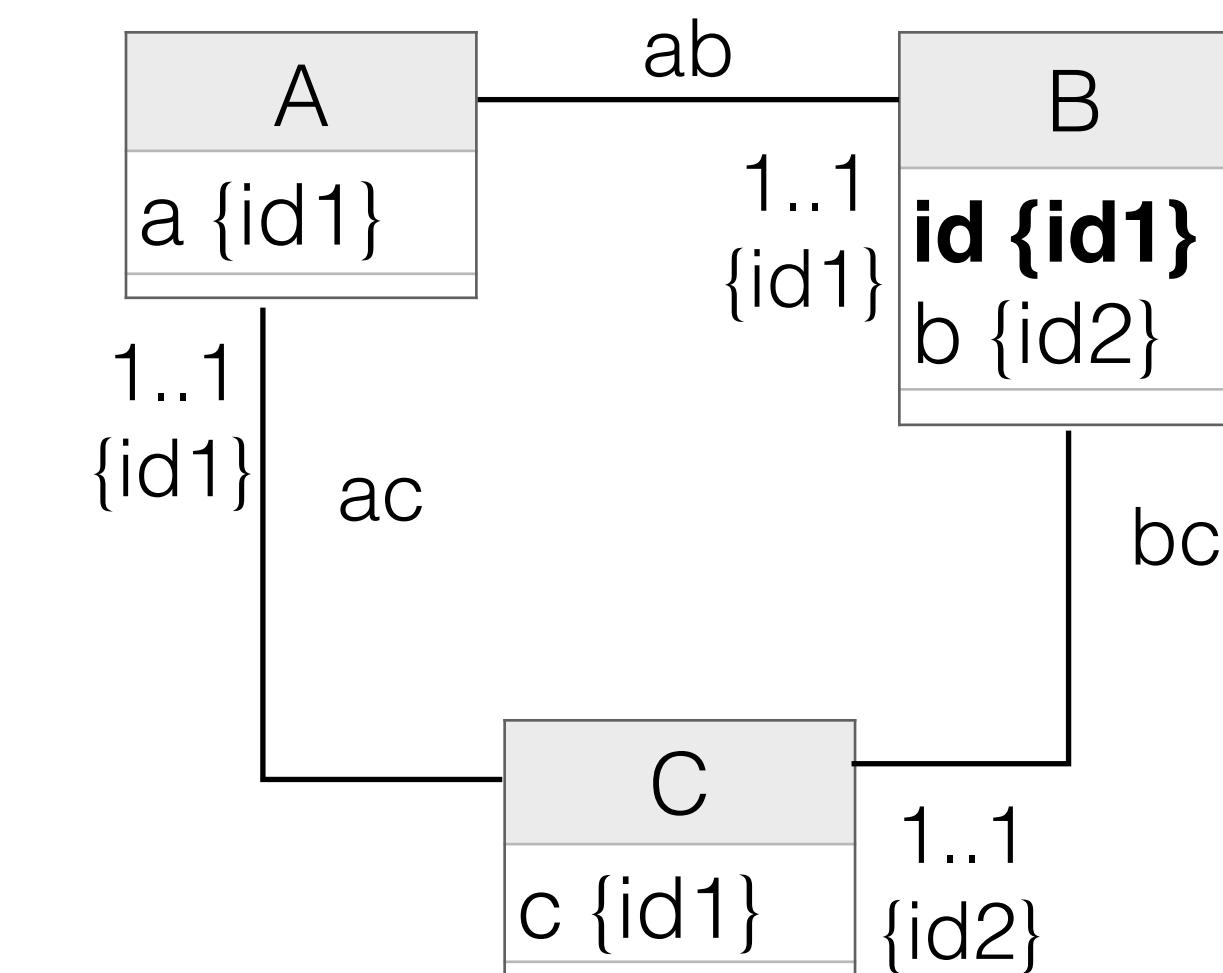
Ciclo di identificatori primari esterni:

- per identificare (id. primario) un oggetto di A serve il valore dell'attributo 'a' e un oggetto di B
- per identificare (id. primario) un oggetto di B serve il valore dell'attributo 'b' e un oggetto di C
- per identificare (id. primario) un oggetto di C serve il valore dell'attributo 'c' e un oggetto di ...A (!)



Rompere cicli di identificatori primari esterni:

- scegliere la/le classi più opportuna/e ed eleggere per questa/e un altro identificatore (anche creandone uno artificiale)





Al termine del passo di definizione degli identificatori di classi, tutte le classi hanno almeno un identificatore, ne hanno certamente uno primario, e gli identificatori primari esterni non formano cicli.

Basi di Dati, Modulo 2

Laurea in Informatica

Parte: Cenni di ingegneria del sw
UNITÀ: B.3

Prof. Toni Mancini
Dipartimento di Informatica



SAPIENZA
UNIVERSITÀ DI ROMA

S.B.3.1

Basi di dati relazionali
Progettazione di basi dati relazionali
Produzione dello schema rel. con vincoli

Ristrutturazione del diagramma UML delle
classi
Specifiche dei vincoli esterni



Obiettivo:

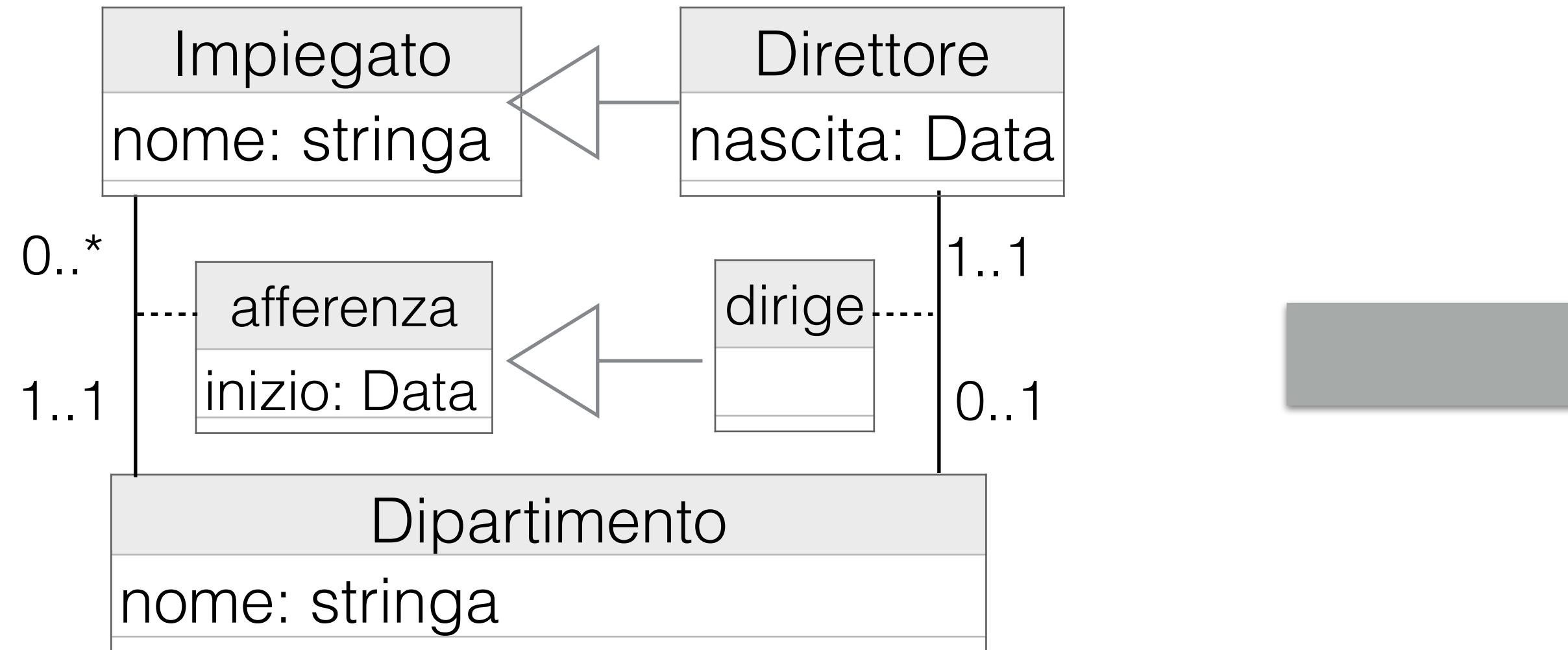
- Ristrutturare i vincoli esterni al diagramma UML concettuale delle classi in vincoli equivalenti sul diagramma UML delle classi ristrutturato.

Metodologia:

- Aggiungere al documento di **specifica ristrutturata dei vincoli esterni** i vincoli esterni definiti, in fase di analisi, nelle specifiche concettuali delle classi e nella specifica concettuale dei vincoli esterni, opportunamente adattati alla struttura del diagramma UML delle classi ristrutturato.

Vincoli esterni (cont.)

Esempio:



Vincolo esterno:

I direttori devono afferire al dipartimento che dirigono da almeno 5 anni.

[V.Dipartimento.direttore_anni_afferenza]

Per ogni oggetto dip:Dipartimento, sia dir:Direttore il direttore di dip, ovvero tale che (dip, dir):dirige.

(Grazie alle due generalizzazioni, è anche vero che (dip, dir):afferenza)

Deve essere: (dir, dip).inizio <= adesso - 5 anni

Vincolo esterno:

[V.Dipartimento.direttore_anni_afferenza]

Per ogni oggetto dip:Dipartimento, sia dir:**Impiegato** il direttore di dip, ovvero tale che (dip, dir):dirige.

Deve essere:

1. **(dir, dip) è anche istanza della associazione “afferenza”**
2. $(\text{dir}, \text{dip}).\text{inizio} \leq \text{adesso} - 5 \text{ anni}$

Basi di Dati, Modulo 2

Laurea in Informatica

Parte: Cenni di ingegneria del sw
UNITÀ: B.3

Prof. Toni Mancini
Dipartimento di Informatica



SAPIENZA
UNIVERSITÀ DI ROMA

S.B.3.1

Basi di dati relazionali
Progettazione di basi dati relazionali
Produzione dello schema rel. con vincoli
Ristrutturazione del diagramma UML delle classi
Specifiche delle operazioni di classe e di use-case

Obiettivo:

- Ristrutturare le specifiche concettuali di classi e di use-case in nuove specifiche equivalenti sul diagramma UML delle classi ristrutturato.

Metodologia:

- Ristrutturare la specifica di ogni operazione di classe, di ogni vincolo di classe, e di ogni operazione di use-case in modo da adattarla alla nuova struttura del diagramma UML delle classi ristrutturato.

Al termine del passo di ristrutturazione della specifica delle operazioni di classe, dei vincoli di classe e delle operazioni di use-case, tutti i documenti di specifica sono ora coerenti con la nuova struttura del diagramma UML delle classi ristrutturato.

Basi di Dati, Modulo 2

Laurea in Informatica

Parte: Cenni di ingegneria del sw
UNITÀ: B.3

Prof. Toni Mancini
Dipartimento di Informatica



SAPIENZA
UNIVERSITÀ DI ROMA

S.B.3.1

Basi di dati relazionali
Progettazione di basi dati relazionali
Produzione dello schema rel. con vincoli

Traduzione diretta del
diagramma UML delle classi
ristrutturato



- **Obiettivo:** generare lo schema relazionale (con vincoli) per la base dati.
 - **Input:** diagramma delle classi UML ristrutturato e specifiche ristrutturate
 - **Output:** un insieme di tavole relazionali con vincoli di integrità.
- **Metodologia:**
 - Ogni classe si traduce in una tabella relazionale
 - Ogni associazione si traduce in una tabella relazionale più vincoli di foreign key (oppure si accorpa nella tabella che traduce una delle classi coinvolte)
 - I vincoli di molteplicità dei ruoli di associazione si traducono in vincoli di chiave, foreign key, o in vincoli esterni (tipicamente, vincoli di inclusione più generale).



Studente
matricola: integer {id1}
nome: varchar
genere: Genere

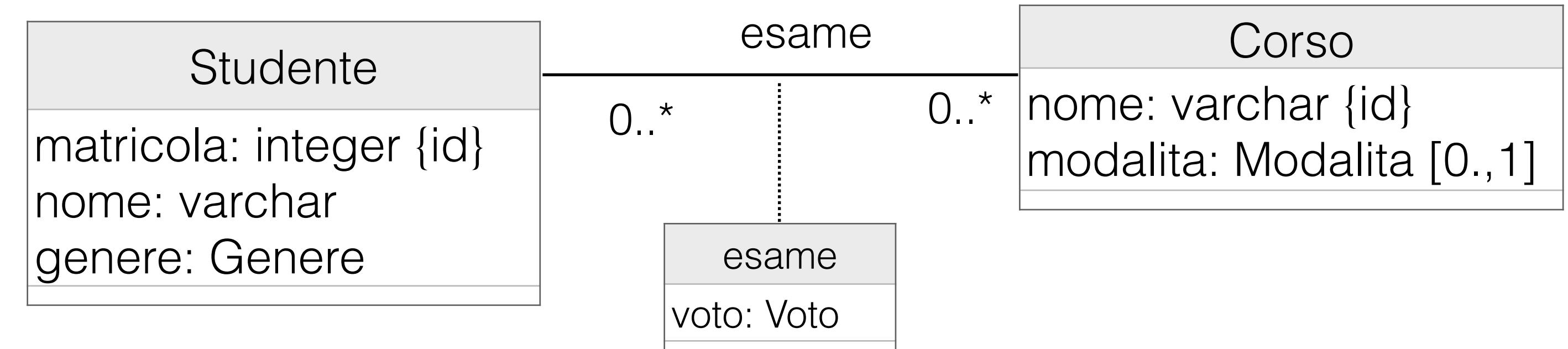
Corso
nome: varchar {id1}
modalita: Modalita [0..1]

Studente(matricola:integer, nome:varchar, genere:Genere)
Corso(nome:varchar, modalita*: Modalita)

```
CREATE TABLE Studente (
    matricola integer not null,
    nome varchar not null,
    genere Genere not null,
    primary key (matricola)
);
```

(*): può assumere valori NULL

```
CREATE TABLE Corso (
    nome varchar not null,
    modalita Modalita, -- anche NULL
    primary key (nome)
);
```



Studente(matricola:integer, nome:varchar, genere:Genere)

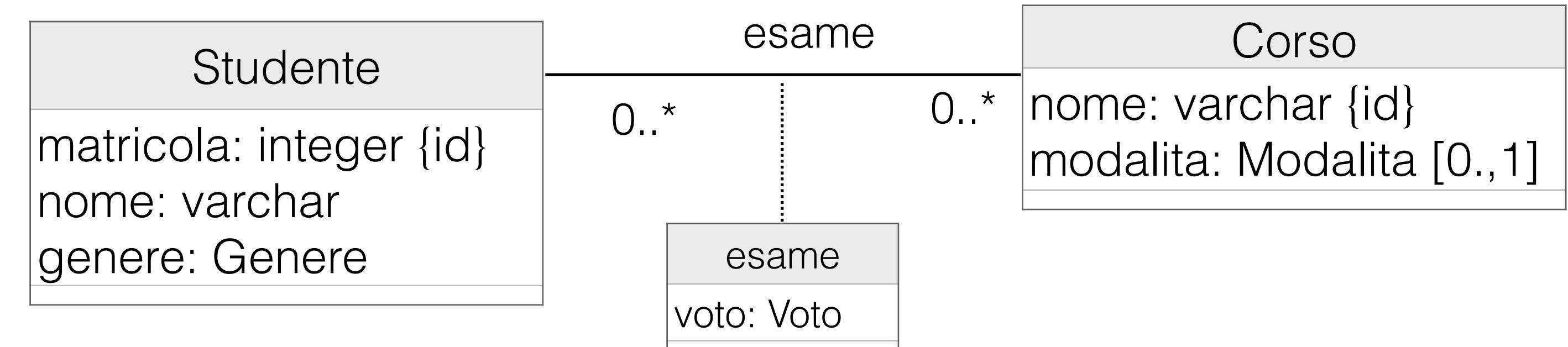
Corso(nome:varchar, modalita* Modalita)

esame (studente:integer , corso:varchar , voto:Voto)

foreign key: studente ref. Studente(matricola)

foreign key: corso ref. Corso(nome)

(*): può assumere valori NULL



Studente(matricola:integer, nome:varchar, genere:Genere)

Corso(nome:varchar, modalita* Modalita)

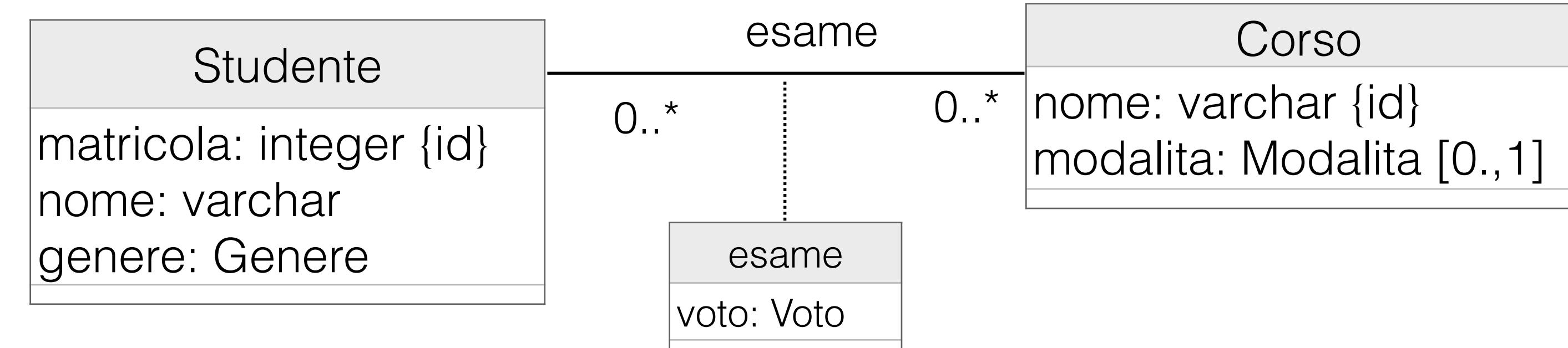
esame (studente:integer corso:varchar, voto:Voto)

foreign key: studente ref. Studente(matricola)

foreign key: corso ref. Corso(nome)

Attributi uno-ad-uno con attributi
di una chiave di Studente

(*): può assumere valori NULL



Studente(matricola:integer, nome:varchar, genere:Genere)

Corso(nome:varchar, modalita* Modalita)

esame (studente:integer | corso:varchar, voto:Voto)

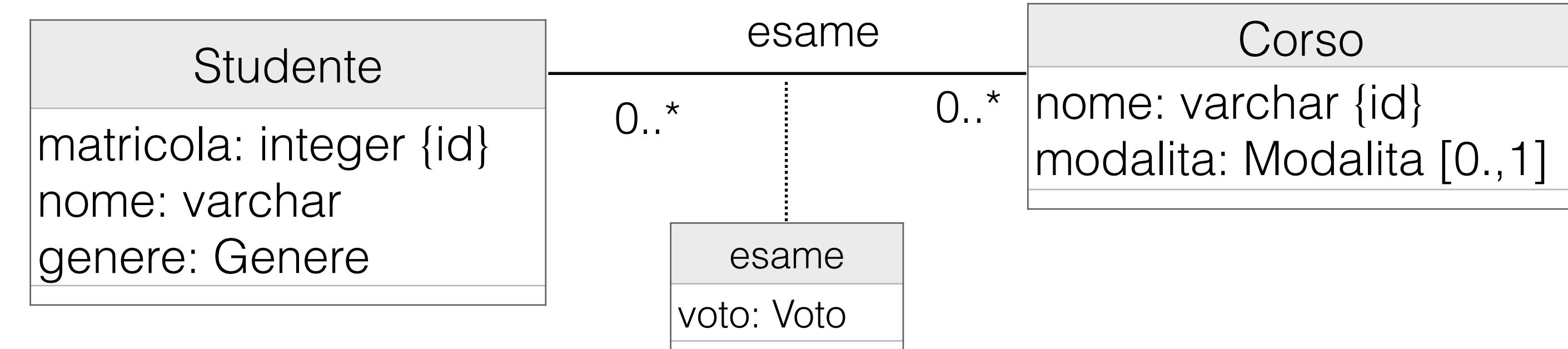
foreign key: studente ref. Studente(matricola)

foreign key: corso ref. Corso(nome)

Attributi uno-ad-uno con attributi
di una chiave di Studente

Attributi uno-ad-uno con
attributi di una chiave di Corso

(*): può assumere valori NULL



Studente(matricola:integer, nome:varchar, genere:Genere)

Corso(nome:varchar, modalita* Modalita)

esame (studente:integer | corso:varchar, voto:Voto)

foreign key: studente ref. Studente(matricola)

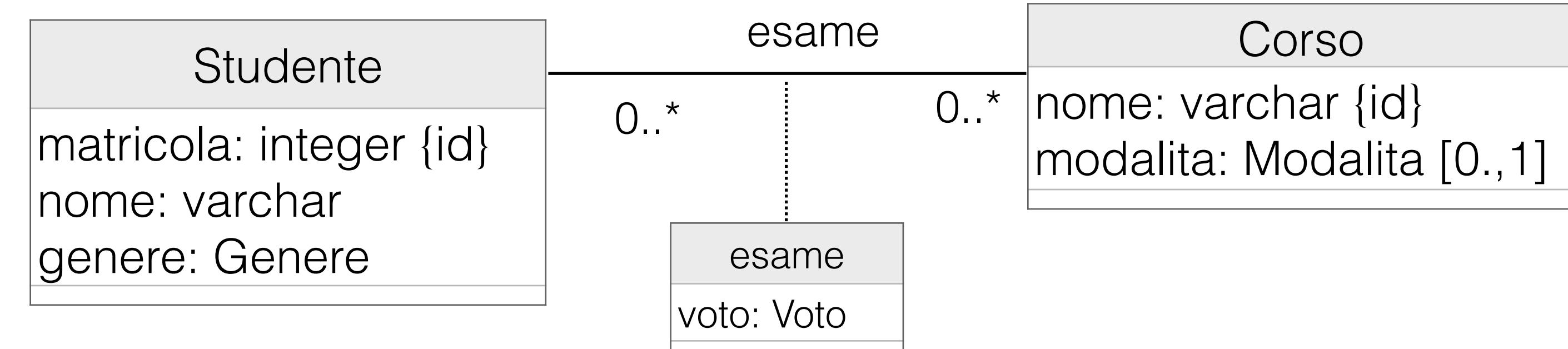
foreign key: corso ref. Corso(nome)

Attributi uno-ad-uno con attributi
di una chiave di Studente

Attributi uno-ad-uno con
attributi di una chiave di Corso

Tutti e soli gli attributi
chiave primaria di
'esame'

(*): può assumere valori NULL



Studente(matricola:integer, nome:varchar, genere:Genere)

Corso(nome:varchar, modalita* Modalita)

esame (studente:integer | corso:varchar, voto:Voto)

foreign key: studente ref. Studente(matricola)

foreign key: corso ref. Corso(nome)

Attributi uno-ad-uno con attributi
di una chiave di Studente

Attributi uno-ad-uno con
attributi di una chiave di Corso

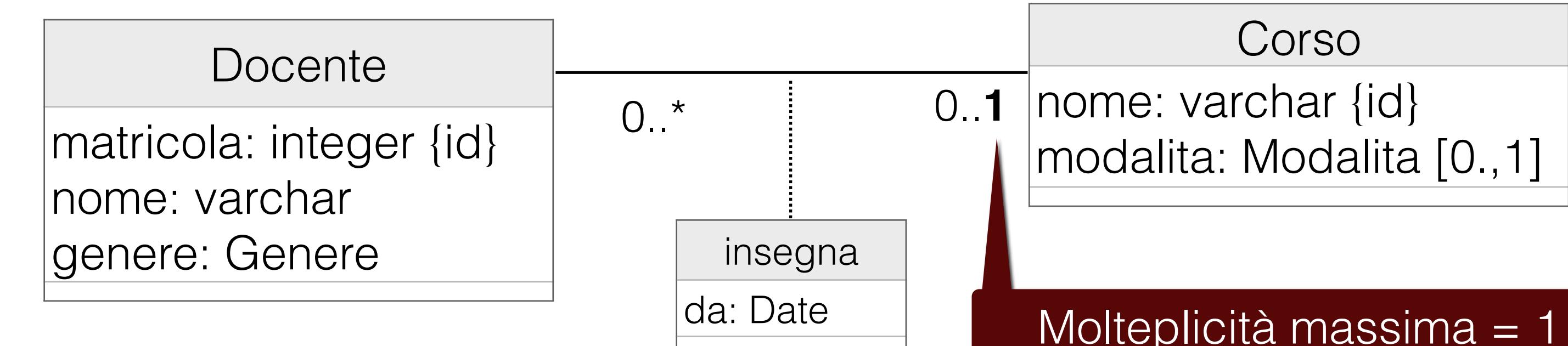
(*): può assumere valori NULL

```

CREATE TABLE esame (
    studente integer not null,
    Non deve essere varchar?
    corso integer not null,
    voto Voto not null,
    primary key (studente, corso),
    foreign key (studente) references
        Studente (matricola),
    foreign key (corso) references
        Corso (nome)
);
  
```

Tutti e soli gli attributi
chiave primaria di
'esame'

Passo 1: Vincoli di molteplicità massima 1



Docente(matricola:integer, nome:varchar, genere:Genere)

Corso(nome:varchar, modalita*: Modalita)

insegna (docente:integer, corso:varchar, da:Date)

altra chiave: docente

foreign key: docente references Docente(matricola)

foreign key: corso references Corso(nome)

Corso

nome: varchar {id}
modalita: Modalita [0..1]

Molteplicità massima = 1
→ Vincolo di integrità in
'insegna'

chiave primaria “inutilmente”
lunga → si accorta

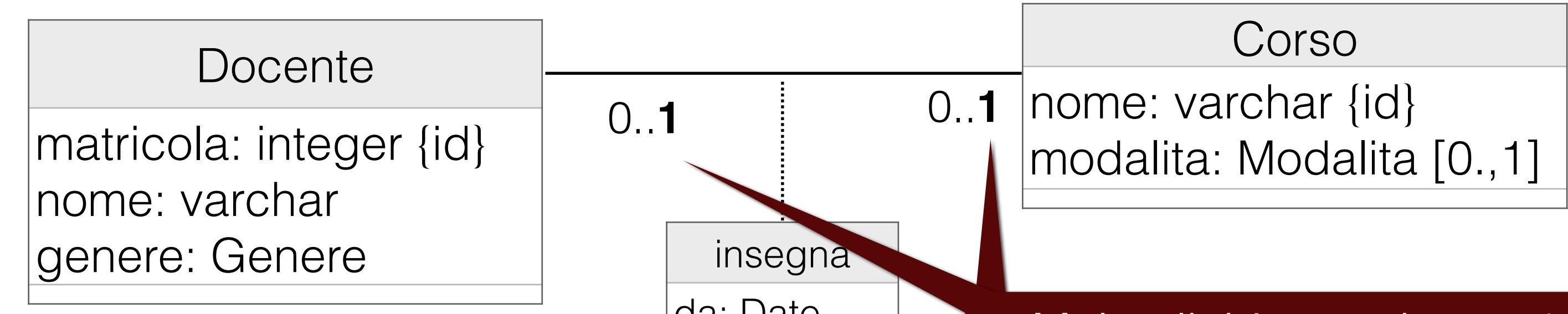
insegna (docente:integer , corso:varchar, da:Date)

altra chiave: docente

foreign key: docente references Docente(matricola)

foreign key: corso references Corso(nome)

Passo 1: Vincoli di molteplicità massima 1 (cont.)



Docente(matricola:integer, nome:varchar, genere:Genere)
Corso(nome:varchar, modalita*: Modalita)

insegna (docente:integer, corso:varchar, da:Date)

altra chiave: corso

altra chiave: docente

foreign key: docente references Docente(matricola)

foreign key: corso references Corso(nome)

Corso
nome: varchar {id}
modalita: Modalita [0..1]

Molteplicità massima = 1
—> Vincolo di integrità in
‘insegna’

chiave primaria “inutilmente” lunga
—> si accorcia... in un modo o nell’altro

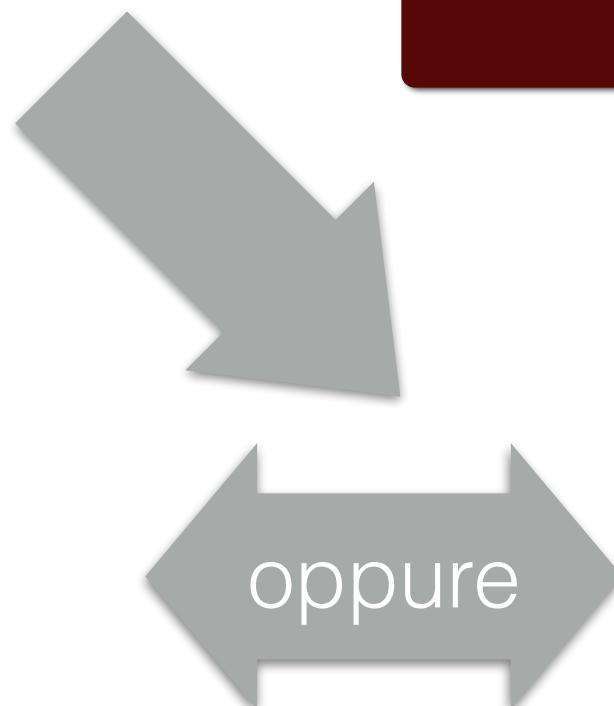
insegna (docente:integer, **corso:varchar**, da:Date)

~~altra chiave: corso~~

~~altra chiave: docente~~

foreign key: docente references Docente(matricola)

foreign key: corso references Corso(nome)



insegna (**docente:integer**, corso:varchar, da:Date)

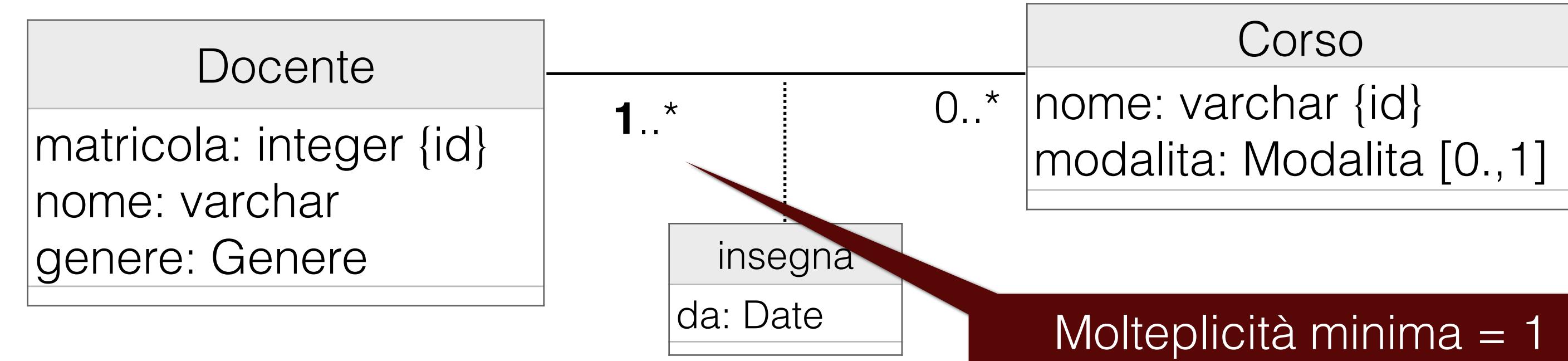
altra chiave: corso

~~altra chiave: docente~~

foreign key: docente references Docente(matricola)

foreign key: corso references Corso(nome)

Passo 2: Vincoli di molteplicità minima 1



Molteplicità minima = 1
→ Vincolo di integrità in
'Corso'

Docente(matricola:integer, nome:varchar, genere:Genere)

Corso(nome:varchar, modalita*:Modalita)

v. inclusione: Corso(nome) occorre in insegna(corso)

insegna (docente:integer, corso:varchar, da:Date)

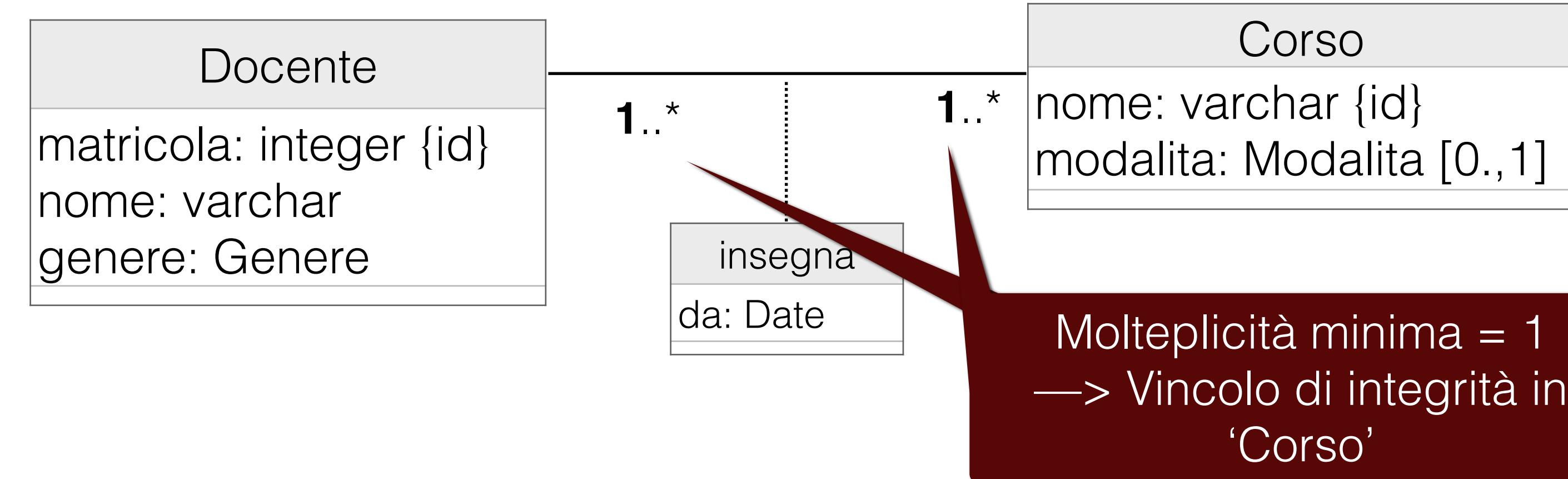
foreign key: docente references Docente(matricola)

foreign key: corso references Corso(nome)

Vincolo di inclusione: generalizzazione di vincolo di foreign key, ma gli attributi della tabella di destinazione **non** formano una chiave.

I DBMS non offrono costrutti che lo implementano direttamente

Passo 2: Vincoli di molteplicità minima 1 (cont.)



Docente(matricola:integer, nome:varchar, genere:Genere)

v. inclusione: Docente(matricola) occorre in insegna(docente)

Corso(nome:varchar, modalita*: Modalita)

v. inclusione: Corso(nome) occorre in insegna(corso)

insegna(docente:integer, corso:varchar, da:Date)

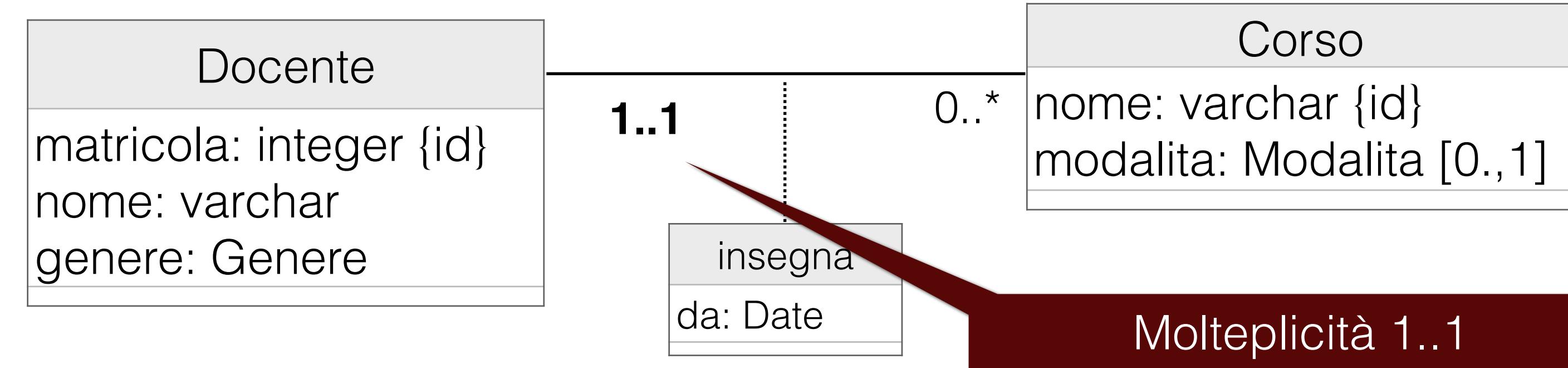
foreign key: docente references Docente(matricola)

foreign key: corso references Corso(nome)

Vincolo di inclusione: generalizzazione di vincolo di foreign key, ma gli attributi della tabella di destinazione **non** formano una chiave.

I DBMS non offrono costrutti che lo implementano direttamente

Caso speciale: Vincoli di molteplicità 1..1



Molteplicità 1..1
—> Combiniamo i due passi
precedenti

Docente(matricola:integer, nome:varchar, genere:Genere)

Corso(nome:varchar, modalita*: Modalita)

v. inclusione: Corso(nome) occorre in insegna(corso)

insegna (docente:integer, corso:varchar, da:Date)

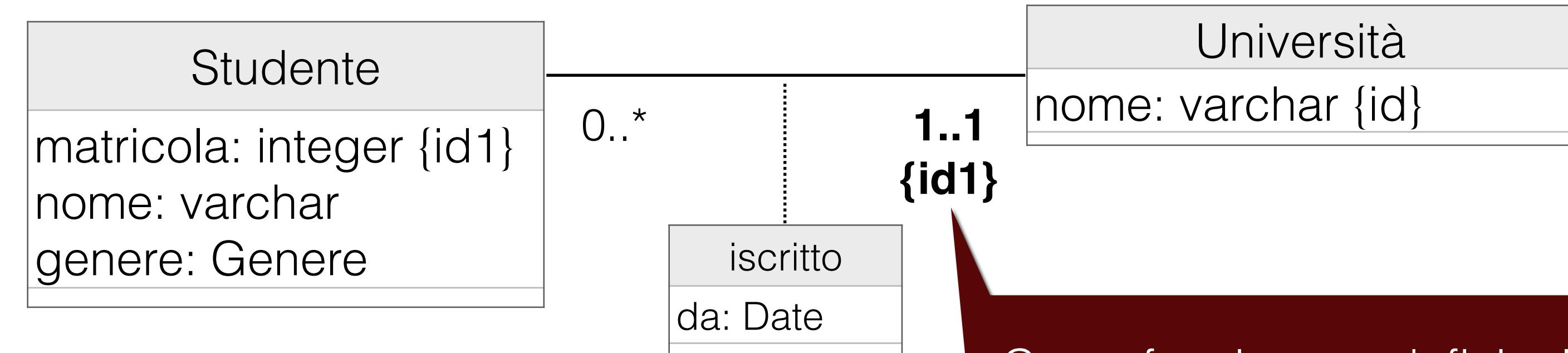
foreign key: docente references Docente(matricola)

foreign key: corso references Corso(nome)

Vincolo di inclusione: in questo caso l'attributo 'corso' forma una chiave completa della tabella Corso.
Il vincolo di inclusione è implementabile tramite un vincolo di foreign key:

foreign key: Corso(nome) references insegna(corso)

L'identificatore primario di Studente coinvolge un ruolo di associazione (ovviamente a molt. 1..1)



Come facciamo a definire la chiave primaria della tabella Persona?

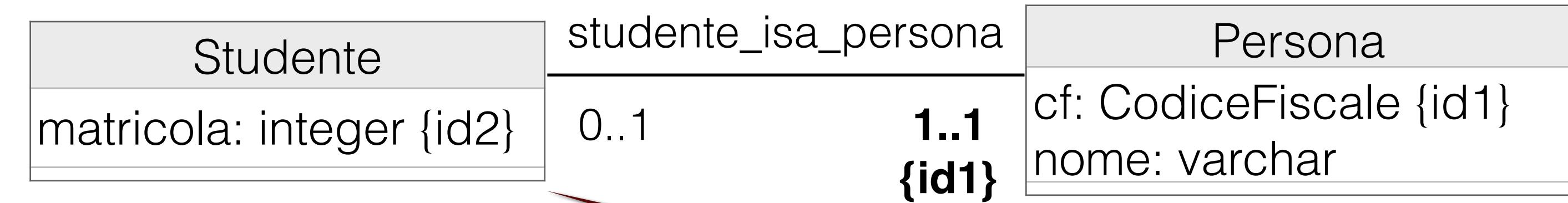
Universita(nome:varchar)

Studente(**matricola:integer**, nome:varchar, genere:Genere,
universita:varchar, iscritto_da:Date)

foreign key: universita references Universita(nome)

Accorpamento: la tabella che implementa l'associazione 'iscritto' viene accorpata nella tabella che implementa la classe Studente (nota: corrispondenza delle righe 1-1)

Caso tipico: associazione che deriva da ristrutturazione di relazione is-a



Come facciamo a definire la chiave primaria della tabella Studente?

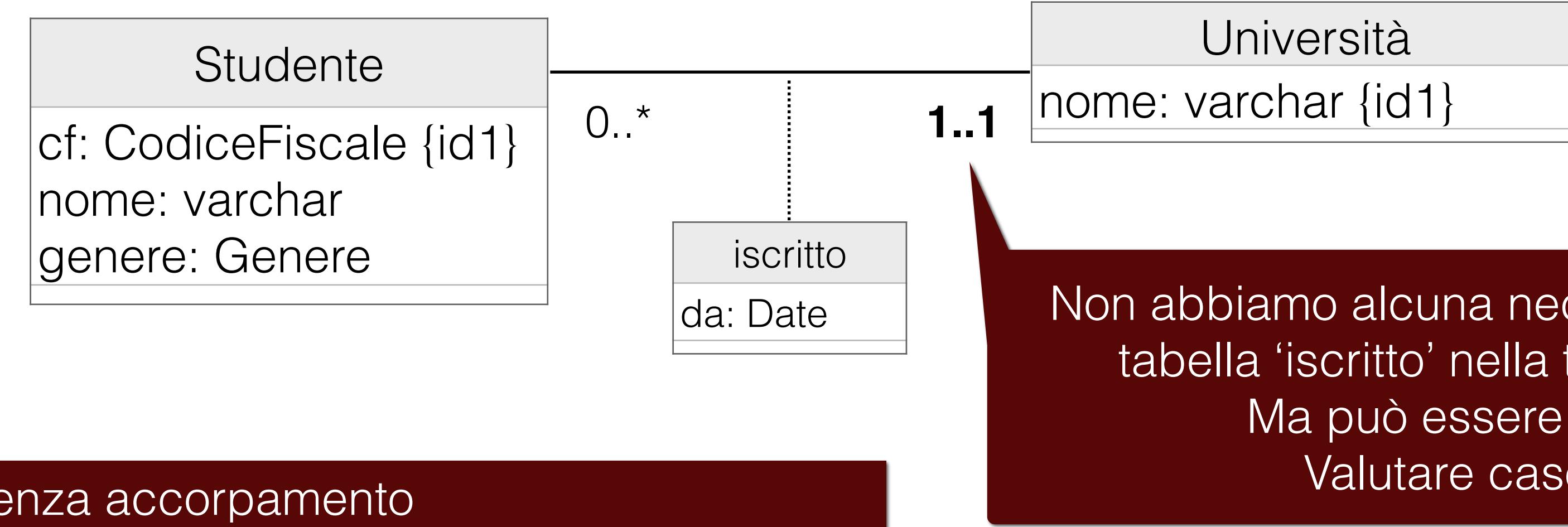
Persona(cf:CodiceFiscale, nome:varchar)

Studente(**persona:CodiceFiscale**, matricola:integer)

altra chiave: matricola

foreign key: persona references Persona(cf)

E' possibile procedere ad accorpamenti ogni volta che un'associazione ha un ruolo a molt. 1..1 o 0..1



Universita(nome:varchar)

Studente(**cf:CodiceFiscale**, nome:varchar, genere:Genere)

v. inclusione: Studente(cf) occorre in iscritto(studente)

—> **foreign key: Studente(cf) references iscritto(studente)**

iscritto(**studente:CodiceFiscale**, universita:varchar)

foreign key: studente references Studente(cf)

foreign key: universita references Universita(nome)

Non abbiamo alcuna necessità di accoppare la tabella 'iscritto' nella tabella 'Studente'...

Ma può essere conveniente.

Valutare caso per caso

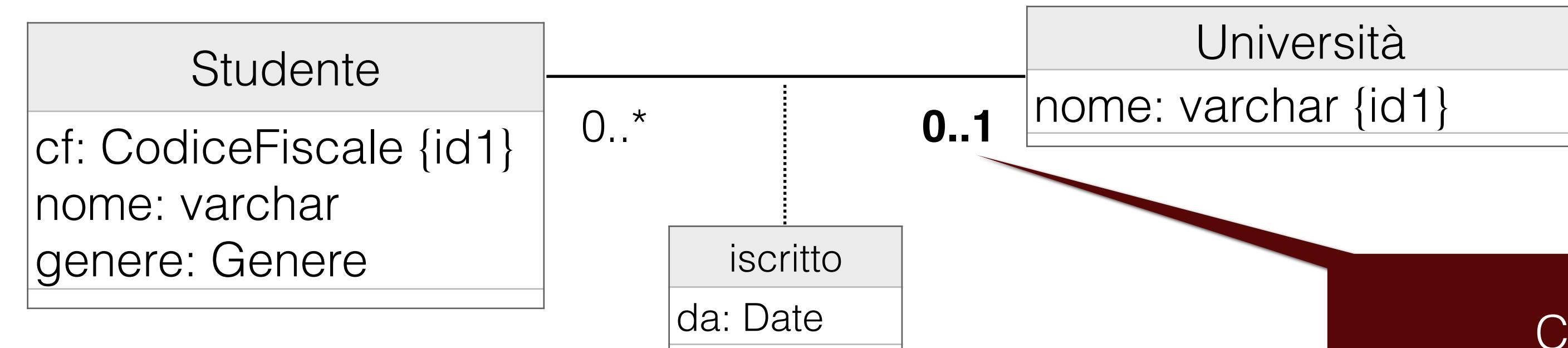
Con accorpamento

Universita(nome:varchar)

Studente(**cf:CodiceFiscale**, nome:varchar, genere:Genere,
universita:varchar, iscritto_da:Date)

foreign key: universita references Universita(nome)

E' possibile procedere ad accorpamenti ogni volta che un'associazione ha un ruolo a molt. 1..1 o 0..1



Senza accorpamento

Universita(nome:varchar)

Studente(**cf:CodiceFiscale**, nome:varchar, genere:Genere)

v. inclusione: Studente(cf) occorre in iscritto(studente)

→ **foreign key: Studente(cf) references iscritto(studente)**

iscritto(**studente:CodiceFiscale**, universita:varchar)

foreign key: studente references Studente(cf)

foreign key: universita references Universita(nome)

Caso a molt. 0..1

Con accorpamento

Universita(nome:varchar)

Studente(**cf:CodiceFiscale**, nome:varchar, genere:Genere,
universita*:varchar, iscritto_da*:Date)

foreign key: universita references Universita(nome)

v. ennupla: (universita is null) = (iscritto_da is null)

(*) può assumere valori NULL

Basi di Dati, Modulo 2

Laurea in Informatica

Parte: Cenni di ingegneria del sw
UNITÀ: B.3
Prof. Toni Mancini
Dipartimento di Informatica



SAPIENZA
UNIVERSITÀ DI ROMA

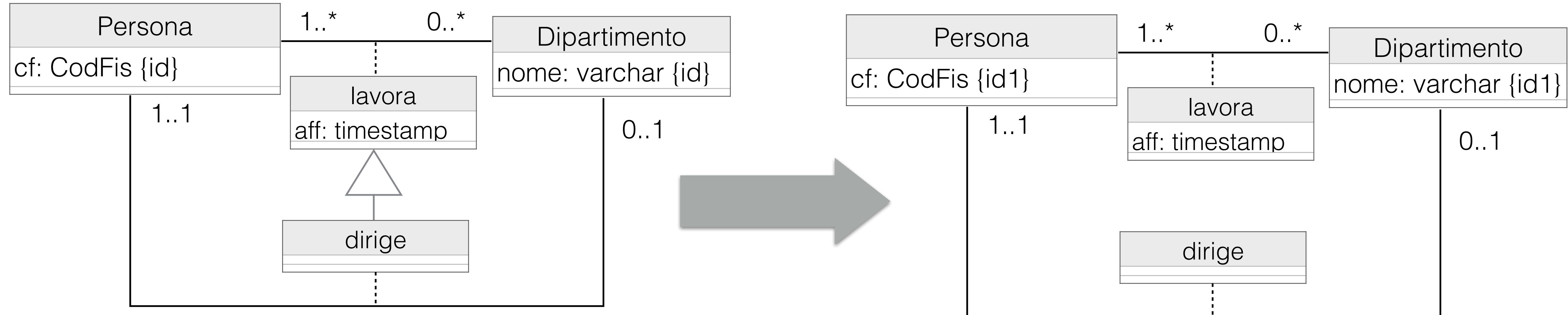
S.B.3.1

Basi di dati relazionali
Progettazione di basi dati relazionali
Produzione dello schema rel. con vincoli

Vincoli esterni

Vincoli esterni allo schema relazionale

- **Obiettivo:** progettare come implementare, nel DBMS, ogni vincolo esterno.
 - **Input:** schema relazionale, vincoli esterni ristrutturati
 - **Output:** decisioni su come implementare i vincoli esterni
- **Metodologia:**
 - Utilizzare, ove possibile, i costrutti di unique, foreign key, e gli altri costrutti per imporre vincoli di integrità supportati dal DBMS in uso
 - Per tutti gli altri, definire opportuni **trigger**
 - **Trigger:** programma imperativo (in linguaggio proprietario del DBMS!) eseguito all'interno del DBMS che:
 - intercetta un evento (insert/update/delete di ennuple in una o più tabella)
 - esegue un controllo (codice imperativo con SQL immerso)
 - può sollevare un'eccezione per rifiutare l'operazione intercettata



Vincolo esterno [V.dirige.isa_lavora]
 $\forall d, p \text{ dirige}(d, p) \rightarrow \text{lavora}(d, p)$

Definizione delle seguenti relazioni con vincoli:

Persona(cf:CodFis)

Dipartimento(id:integer, nome:varchar)

VincoloDB: inclusione: $\text{id} \subseteq \text{dirige}(\text{dip})$

VincoloDB: inclusione: $\text{id} \subseteq \text{lavora}(\text{dip})$

Può essere ignorato, perché
è implicato dagli altri

Implementable come
foreign key, perché
dirige(dip) è chiave

lavora(dip:integer, persona:CodFis)

VincoloDB: foreign key: persona references Persona(cf)

VincoloDB: foreign key: dip references Dipartimento(id)

dirige(dip:integer, persona:CodFis)

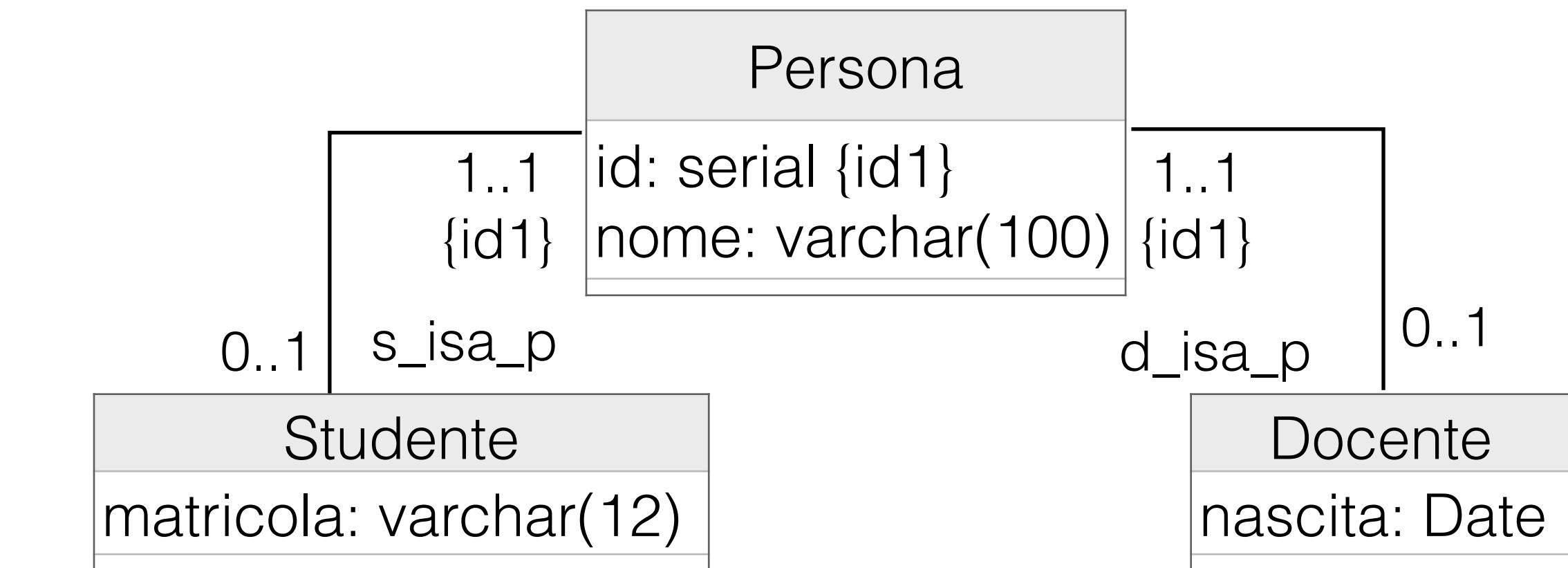
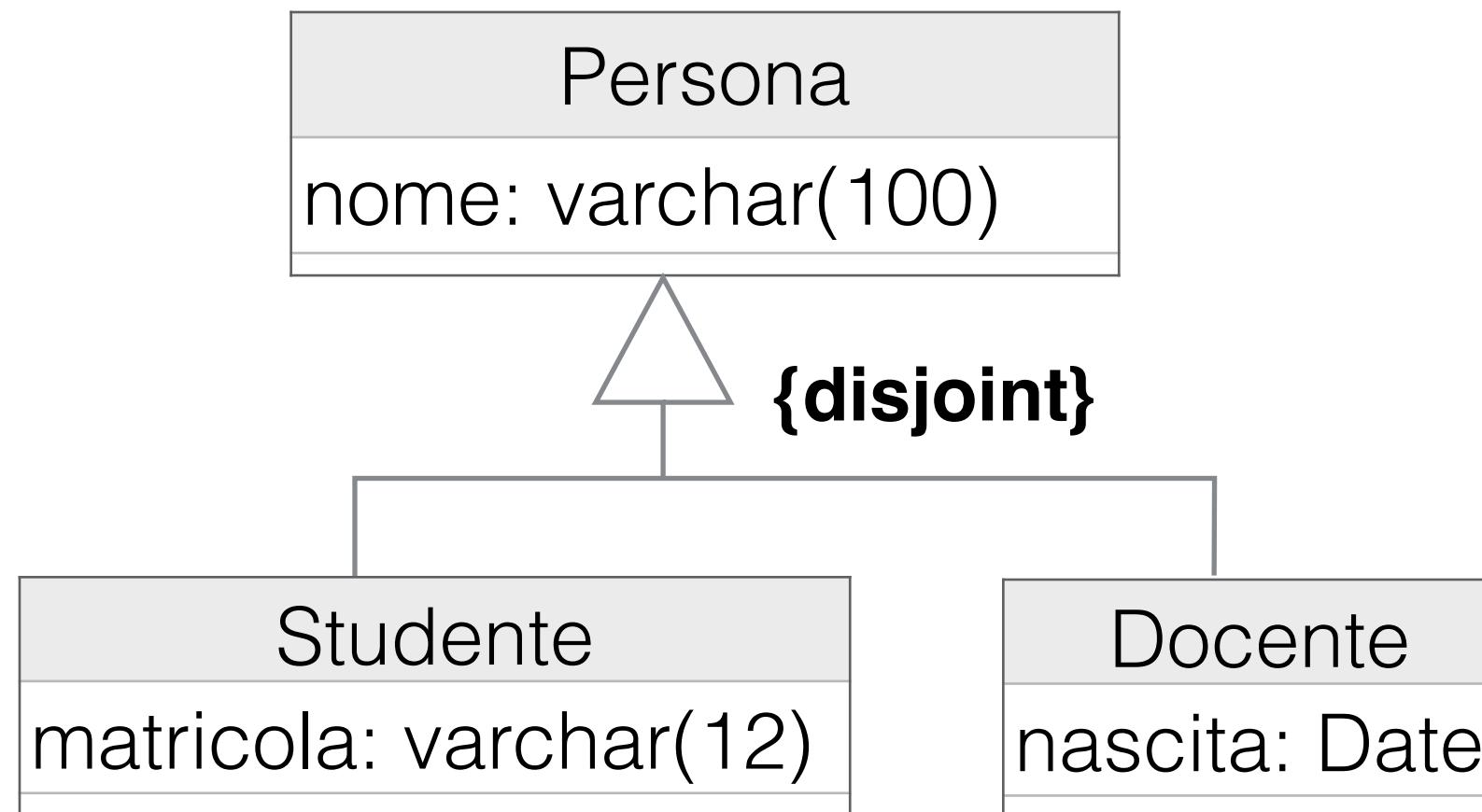
VincoloDB: inclusione: $(\text{dip}, \text{persona}) \subseteq \text{lavora}(\text{dip}, \text{persona})$

VincoloDB chiave: persona

Implementabile come foreign key,
perché **lavora(dip, persona)** è chiave

Esempio 1: vincoli esterni allo schema relazionale

Come implementare invece questo vincolo esterno?



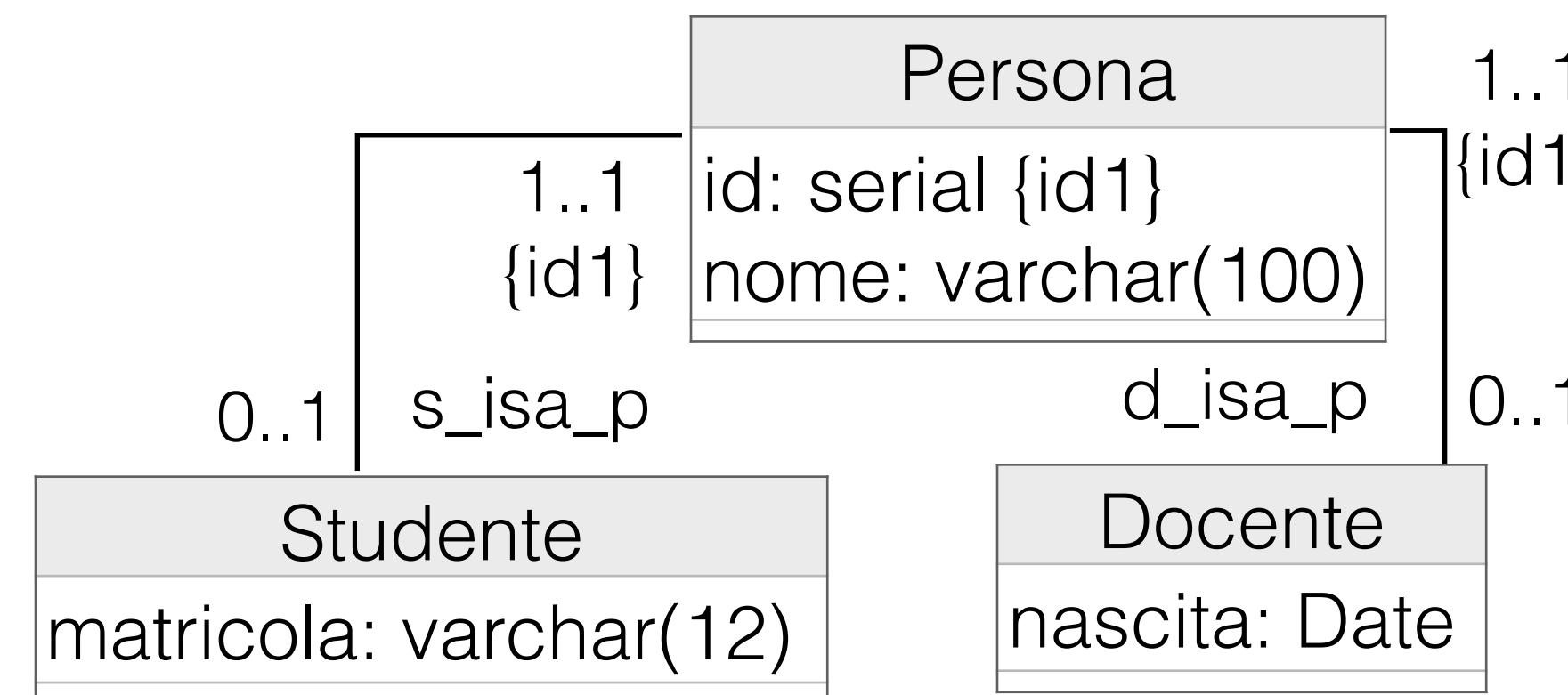
Vincolo esterno [V.Persona.gen.disj]

Non devono esistere p :Persona, s :Studente e d :Docente tali che
 $(s, p): s_isa_p$ e $(d, p): d_isa_p$:

$$\neg \exists p, s, d \text{ Persona}(p) \wedge \text{Studente}(s) \wedge \text{Docente}(d) \wedge s_isa_p(s, p) \wedge d_isa_p(d, p)$$

Non è implementabile tramite primary key, unique, foreign key, check (il costrutto check permette la definizione di soli vincoli di ennupla)

Esempio 1: progettazione di vincoli esterni mediante trigger



- Persona(id:serial, nome:varchar(100))
- Studente(persona:integer, matricola:varchar(12)) // accorpa s_isap foreign key persona references Persona(id)
- Docente(persona:integer, nascita:date) // accorpa d_isap foreign key persona references Persona(id)

Progetto del trigger V.Persona.gen.disj

Vincolo esterno [V.Persona.gen.disj]

$\neg \exists p, s, d \text{ Persona}(p) \wedge \text{Studente}(s) \wedge \text{Docente}(d) \wedge s_isa_p(s, p) \wedge d_isa_p(d, p)$

Quali sono gli eventi che possono portare un DB legale a diventare illegale?

Quando invocare la funzione? Appena prima o appena dopo aver effettuato l'operazione (su DB prima o dopo la modifica).

Pseudocodice della funzione con SQL immerso. Argomenti:

- old (solo per UPDATE e DELETE): tupla da cancellare/cancellata o da modificare/prima della modifica
- new (solo per INSERT e UPDATE): tupla da inserire/inserita o a valle della modifica

Operazioni da intercettare: INSERT o UPDATE in Studente o Docente

Quando invocare la funzione: dopo l'operazione intercettata

Funzione(old, new):

- Sia isError = EXISTS(


```

SELECT *
FROM Studente s, Docente d
WHERE s.persona = new.persona
AND d.persona = new.persona
      
```

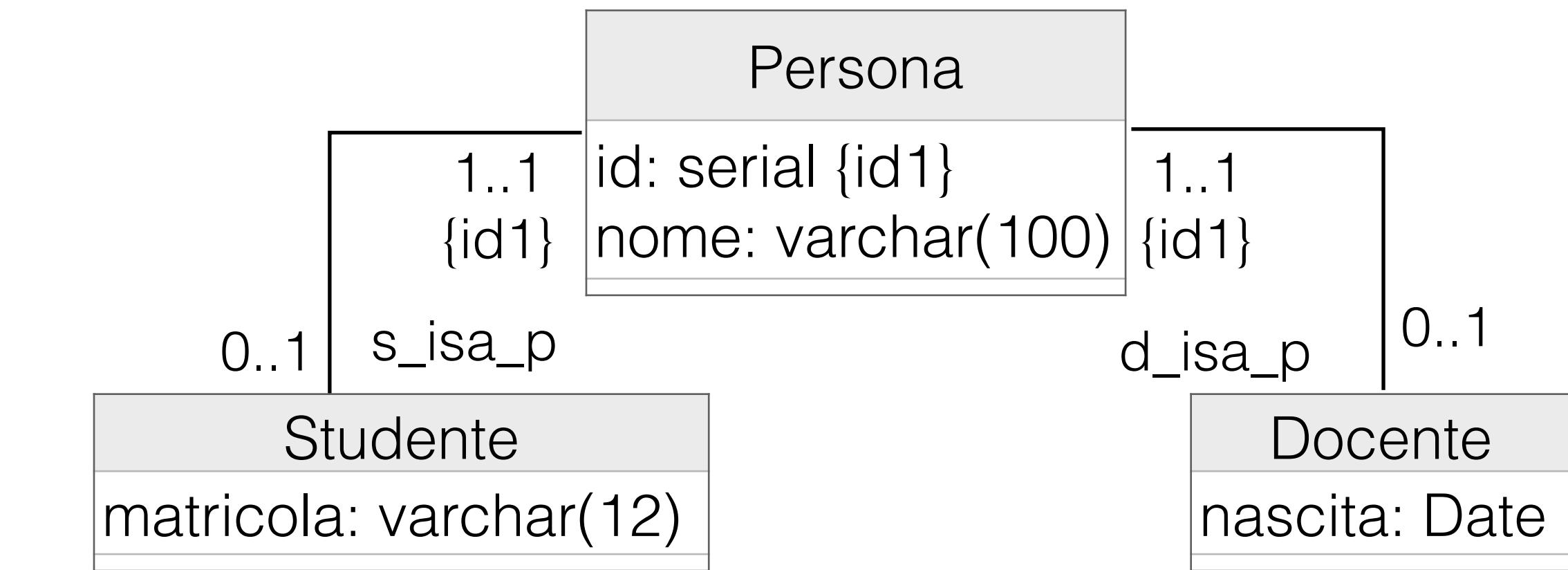
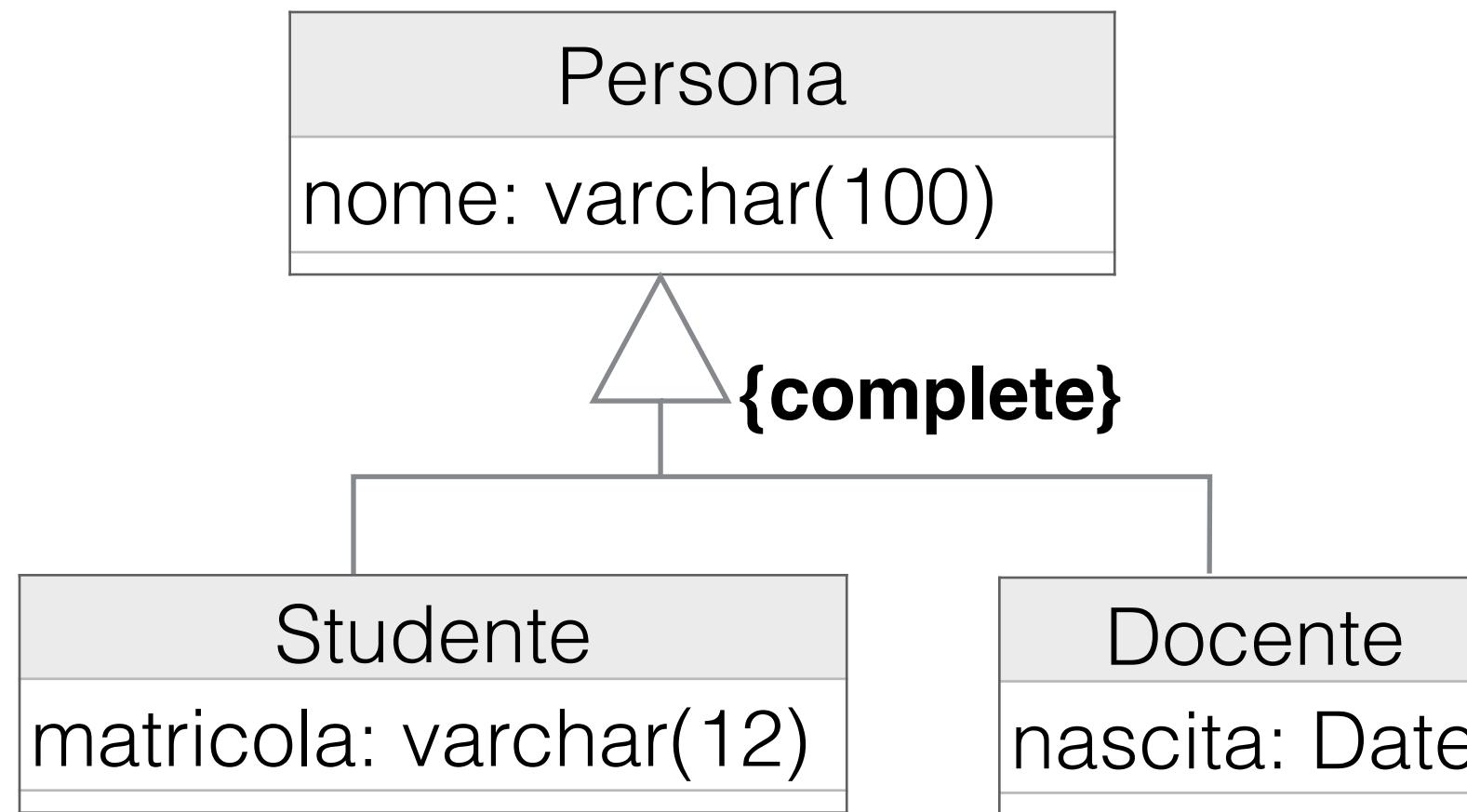
)
- Se isError = TRUE solleva eccezione “INSERT/UPDATE viola il vincolo”
- Altrimenti, permetti l'operazione

perche' non e' {complete}

Perché non vanno intercettati i tentativi di INSERT o UPDATE in Persona?

Esempio 2: vincoli esterni allo schema relazionale

Come implementare invece questo vincolo esterno?



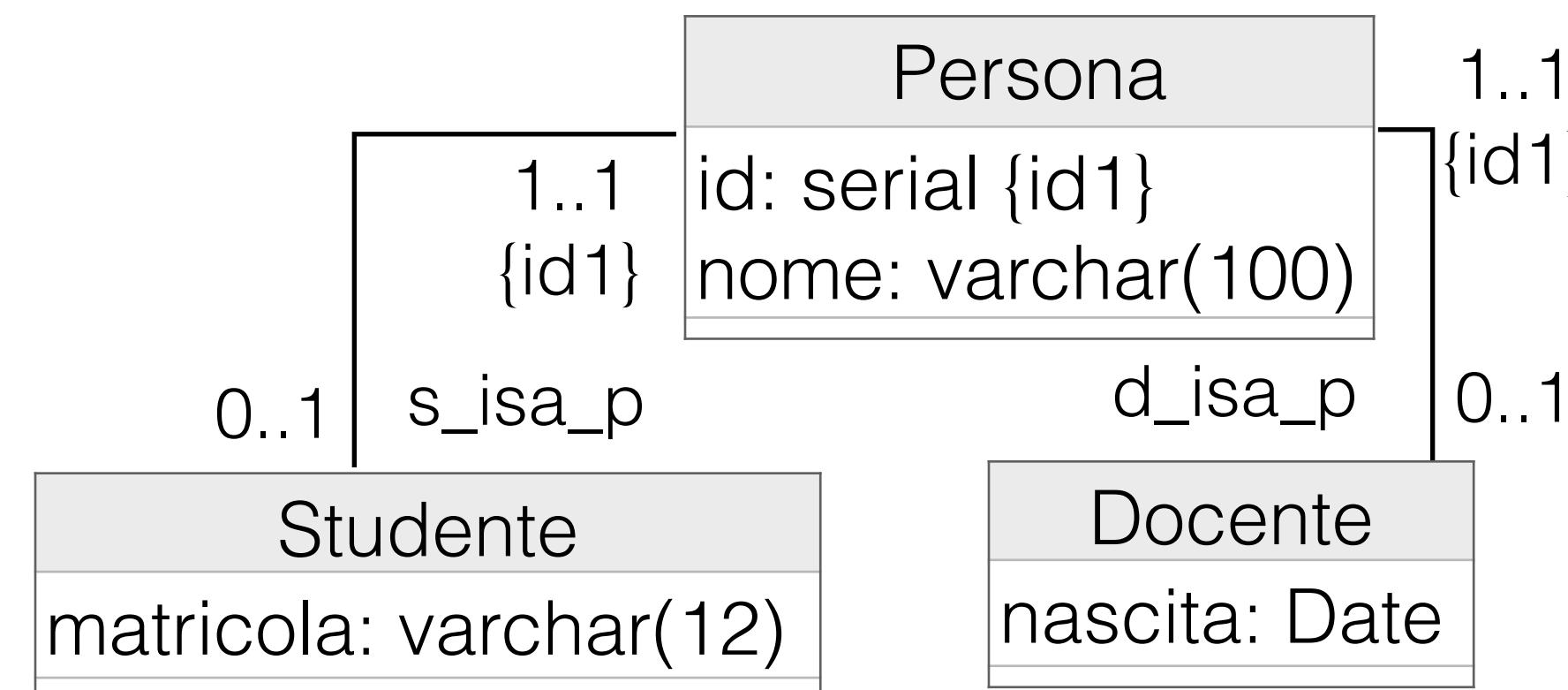
Vincolo esterno [V.Persona.gen.compl]

Per ogni p :Persona:

- esiste s :Studente per cui (s, p) : s_{isa_p} oppure
- esiste d :Docente per cui (d, p) : d_{isa_p}

$$\forall p \text{ Persona}(p) \rightarrow [\exists x s_{isa_p}(x, p) \vee d_{isa_p}(x, p)]$$

Non è implementabile tramite primary key, unique, foreign key, check (il costrutto check permette la definizione di soli vincoli di ennupla)



Vincolo esterno [V.Persona.gen.compl]

$$\forall p \text{ Persona}(p) \rightarrow [\exists x \text{ s_isa}_p(x, p) \vee \text{d_isa}_p(x, p)]$$

Schema relazionale della base dati

- Persona(id:serial, nome:varchar(100))
- Studente(persona:integer, matricola:varchar(12))
 - // accorpa s_isa_p
 - foreign key persona references Persona(id)
- Docente(persona:integer, nascita:date)
 - // accorpa d_isa_p
 - foreign key persona references Persona(id)

Una strategia più avanzata per il progetto del vincolo V.Persona.compl

1. **Politiche di accesso ai dati:** non permettere ad alcun utente di cambiare gli id artificiali
2. **Trigger V_Persona_gen_compl_1_persona:**
 - quando deve scattare: subito dopo INSERT(new) in Persona
 - controllo da effettuare: new.id deve occorrere in Studente(persona) o Docente(persona):
se NOT EXISTS (


```
SELECT persona FROM Studente WHERE persona = new.id
UNION
SELECT persona FROM Docente WHERE persona = new.id
```

) solleva eccezione, altrimenti permetti l'operazione.
3. **Trigger V_Persona_gen_compl_2_[studente I docente]:**
 - quando deve scattare: subito dopo DELETE(old) in Studente o Docente
 - controllo da effettuare: se old.id è in Persona, deve occorrere in Docente(persona) o Studente(persona)
se EXISTS(

```
SELECT *
```

FROM (Persona p LEFT OUTER JOIN Studente s ON p.id = s.persona)
LEFT OUTER JOIN Docente d ON p.id = d.persona
WHERE p.id = old.persona
AND (s.persona IS NULL) AND (d.persona IS NULL)
) allora solleva eccezione, altrimenti permetti l'operazione.

Nota: grazie ad 1., non dobbiamo intercettare gli update (innocui).



Implementazione:

- 1. Politiche di accesso ai dati: non permettere ad alcun utente di cambiare gli id artificiali

```
REVOKE UPDATE (id) ON Persona FROM PUBLIC;  
REVOKE UPDATE (persona) ON Studente FROM PUBLIC;  
REVOKE UPDATE (persona) ON Docente FROM PUBLIC;
```



Implementazione:

-- 2. Trigger compl_1 per INSERT into Persona

```
create or replace function V_Persona_gen_compl_1() returns trigger as $V_Persona_gen_compl_1$
```

-- la funzione vedrà 'new' come argomento

```
declare isError boolean := false;
```

```
begin
```

```
    isError = NOT EXISTS (
```

```
        SELECT persona FROM Studente WHERE persona = new.id
```

```
        UNION ALL
```

```
        SELECT persona FROM Docente WHERE persona = new.id
```

```
);
```

```
if (isError) then raise exception 'Vincolo V_Persona_gen_compl_1 violato (persona.id=%)', new.id;
```

```
end if;
```

return new; -- lascia passare la tupla; in realtà il valore di ritorno per i trigger 'after' è ignorato, non lo sarebbe per i trigger 'before'

```
end; $V_Persona_gen_compl_1$ language plpgsql;
```

```
create constraint trigger V_Persona_gen_compl_1_persona
```

```
after insert on Persona
```

```
deferrable -- affinché sia deferrable, il trigger deve essere di tipo 'constraint'
```

```
for each row execute procedure V_Persona_gen_compl_1();
```



Implementazione:

-- 3. Trigger compl_2 per DELETE from Studente or Docente

create or replace function V_Persona_gen_compl_2() returns trigger as \$V_Persona_gen_compl_2\$

-- la funzione vedrà 'old' come argomento

declare isError boolean := false;

begin

isError := EXISTS(

 SELECT * FROM (Persona p LEFT OUTER JOIN Studente s ON p.id = s.persona) LEFT OUTER JOIN Docente d ON p.id = d.persona
 WHERE p.id = old.persona AND (s.persona IS NULL) AND (d.persona IS NULL)

);

if (isError) then raise exception 'Vincolo V_Persona_isa_compl_2 violato (persona=%)', old.persona;

end if;

return old; — il valore di ritorno per i trigger ‘after’ è ignorato, non lo è per i trigger ‘before’

end;\$V_Persona_gen_compl_2\$ language plpgsql;

create constraint trigger V_Persona_gen_compl_2_studente

after delete on Studente

deferrable for each row execute procedure V_Persona_gen_compl_2();

create constraint trigger V_Persona_gen_compl_2_docente

after delete on Docente

deferrable for each row execute procedure V_Persona_gen_compl_2();



Test:

Data la presenza dell'attributo id:serial in Persona, abbiamo bisogno di ottenere il valore per 'id' generato dal DBMS all'atto dell'inserimento di una ennupla in Persona (INSERT ... RETURNING) e di salvarlo in una variabile locale.

Questo può essere fatto usando un blocco di codice SQL anonimo:

```
begin transaction;  
DO $$ — blocco di codice anonimo (senza dover creare una FUNCTION)  
declare _id integer;  
begin  
    insert into Persona(nome) values ('Carlo') returning id into _id;  
    insert into Studente(persona, matricola) values (_id, 'uni-carlo-00123');  
end$$;  
commit;
```

Esercizio: provare ad effettuare inserimenti o cancellazioni che violerebbero i vincoli.



Esercizio:

Valutare attentamente come ottimizzare la strategia di controllo dei vincoli esterni dei progetti svolti, sfruttando il combinato disposto di:

- Restrizioni nella possibilità di modifiche ai dati, mediante politiche di accesso (REVOKE)
- Trigger che utilizzano function comuni (minimizzando il codice da scrivere)

In particolare, il caso frequente di generalizzazione **disjoint & complete** si presta ad interessanti ottimizzazioni (da scoprire!)

Osservazione 1: possiamo valutare l'opportunità di ridurre i gradi di libertà degli utenti quando inseriscono o modificano ennuple. Ad esempio, potrebbe essere ragionevole:

1. impedire l'update di alcuni attributi (ad es., gli id artificiali, come abbiamo visto)
2. impedire del tutto alcuni UPDATE, forzando gli utenti ad eseguire DELETE + INSERT (per operazioni non critiche)

Osservazione 2: un sistema che implementa i vincoli esterni basandosi sulle restrizioni delle politiche di accesso è intrinsecamente **più fragile** di uno che garantisce che il vincolo sia soddisfatto a valle di qualsivoglia evento, anche quelli impediti (spesso è complicato impedire alcune operazioni a tutti gli utenti, anche agli amministratori). È bene che il progettista valuti **caso per caso**.