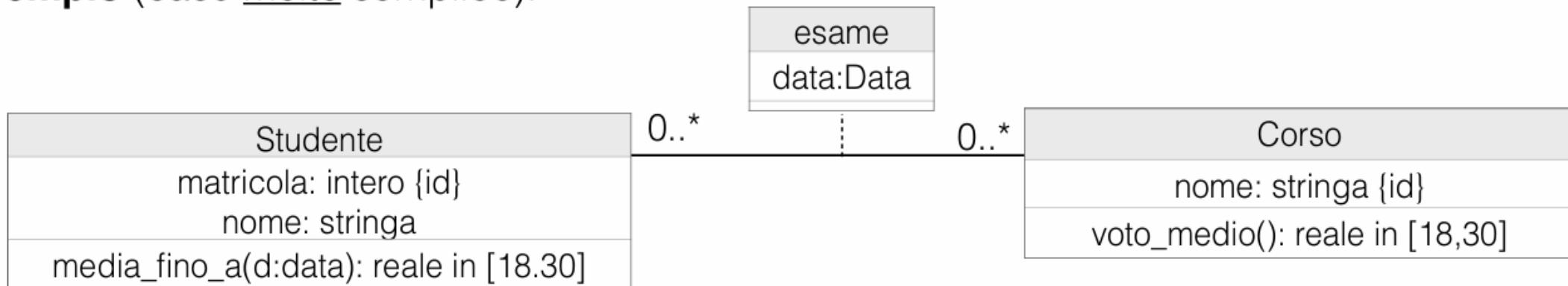
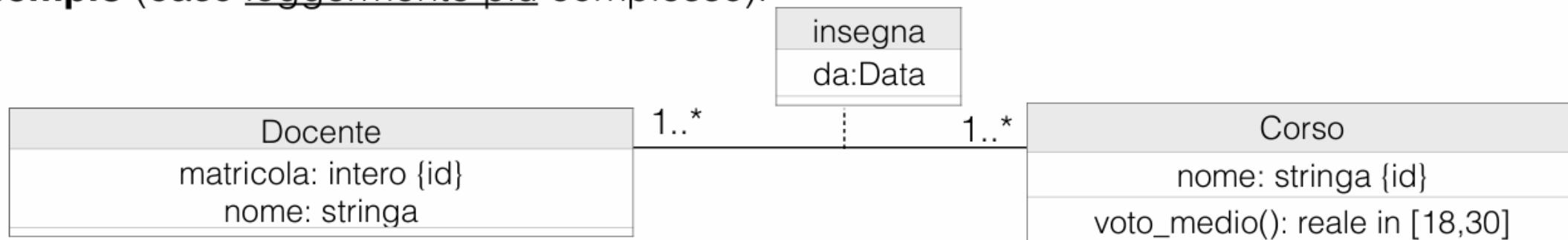


- **Esempio** (caso molto semplice):



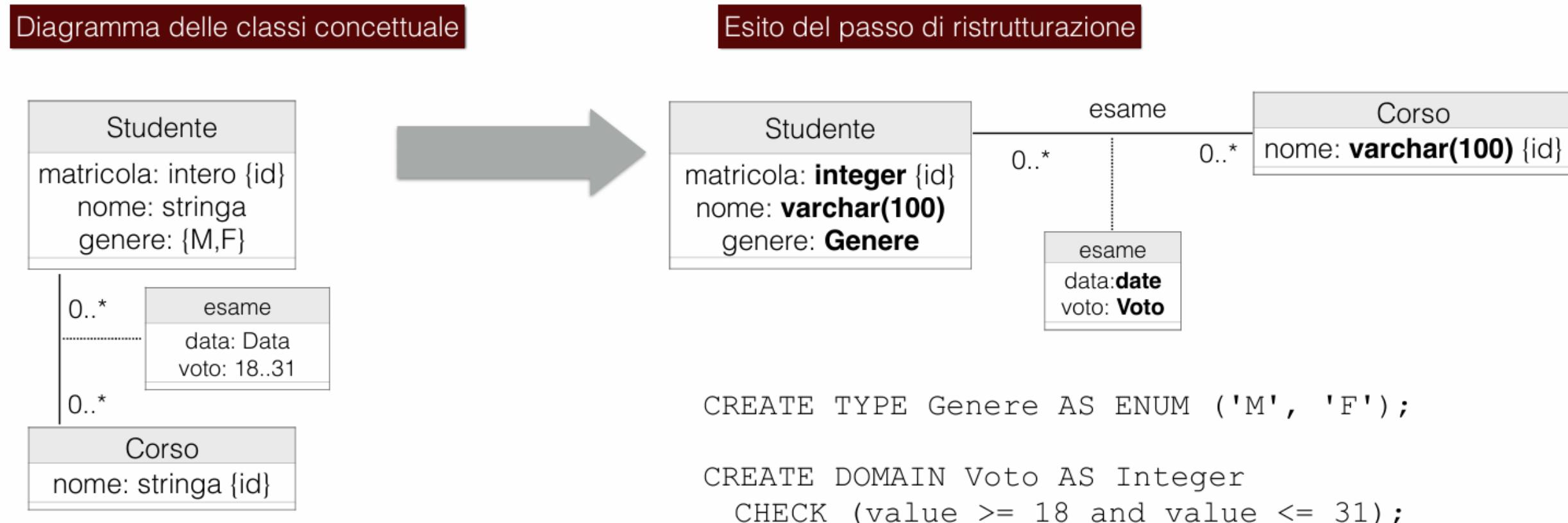
- **Il progettista decide** che le istanze di **Studente**, **Corso** ed **esame** devono essere memorizzate in un DB
- **Il progettista decide** che ogni istanza di classe o associazione sia memorizzata come una singola tupla di una tabella
- **Il progettista definisce** il seguente schema relazionale 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):



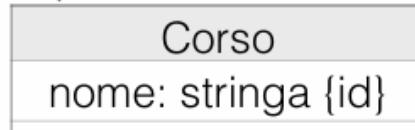
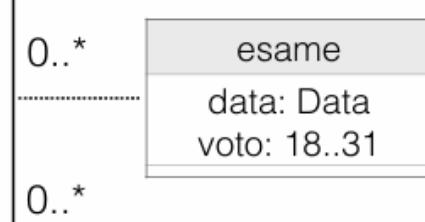
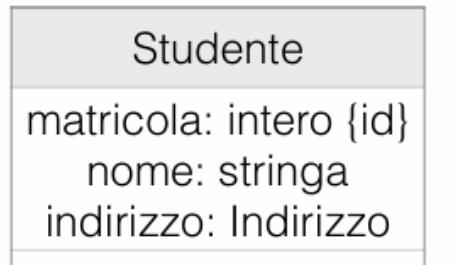
- **Il progettista decide** che le istanze di Docente, Corso ed insegna devono essere memorizzate in un DB
- **Il progettista decide** che ogni istanza di classe o associazione sia memorizzata come una singola ennupla di una tabella
- **Il progettista definisce** il seguente schema relazionale con vincoli 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)

Esempio (tipi base, specializzati, enumerativi):

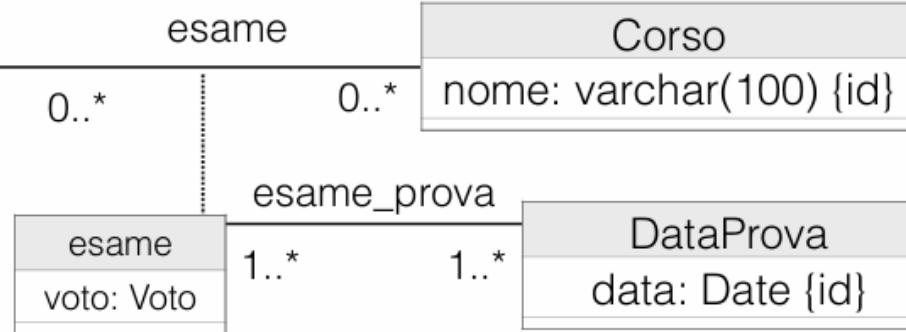


Esempio (tipi composti):

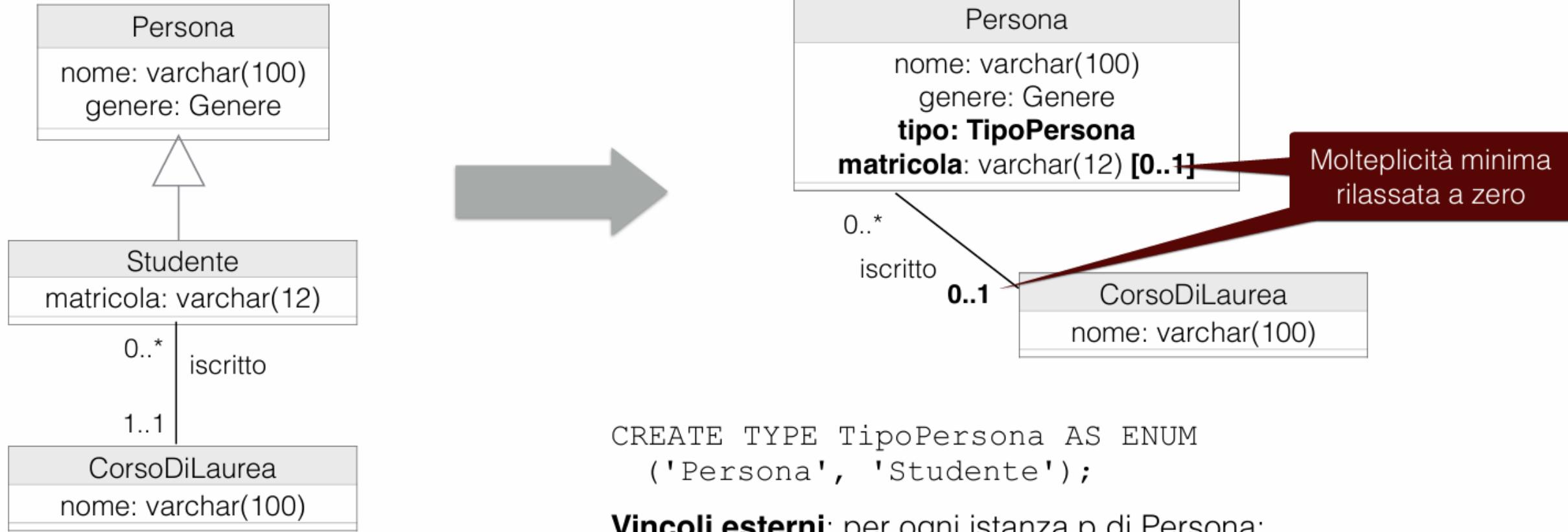
Diagramma delle classi concettuale



Esito del passo di ristrutturazione



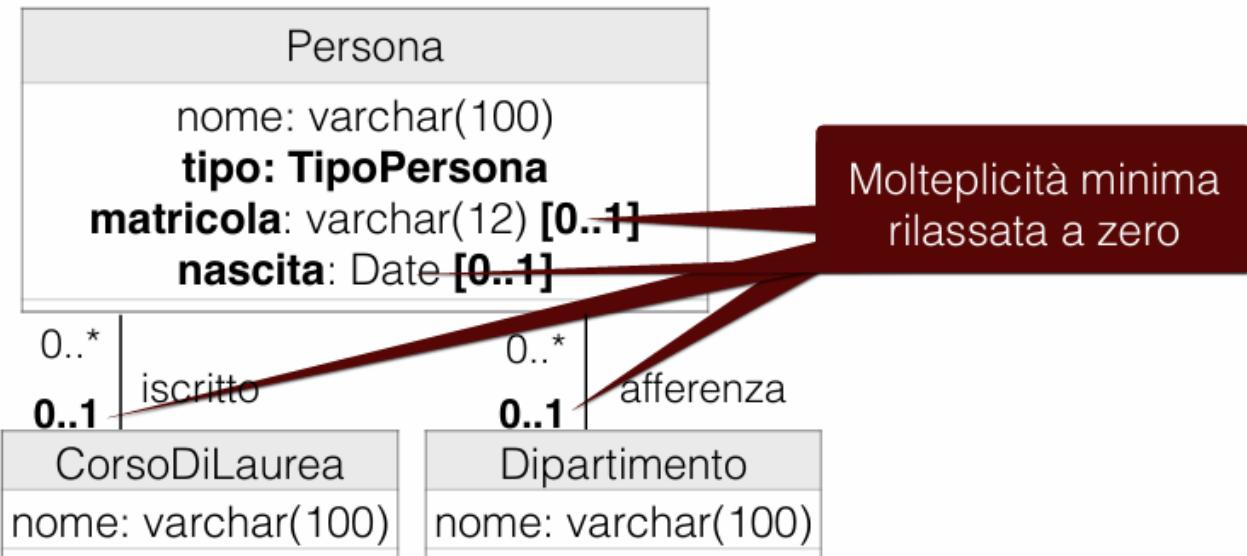
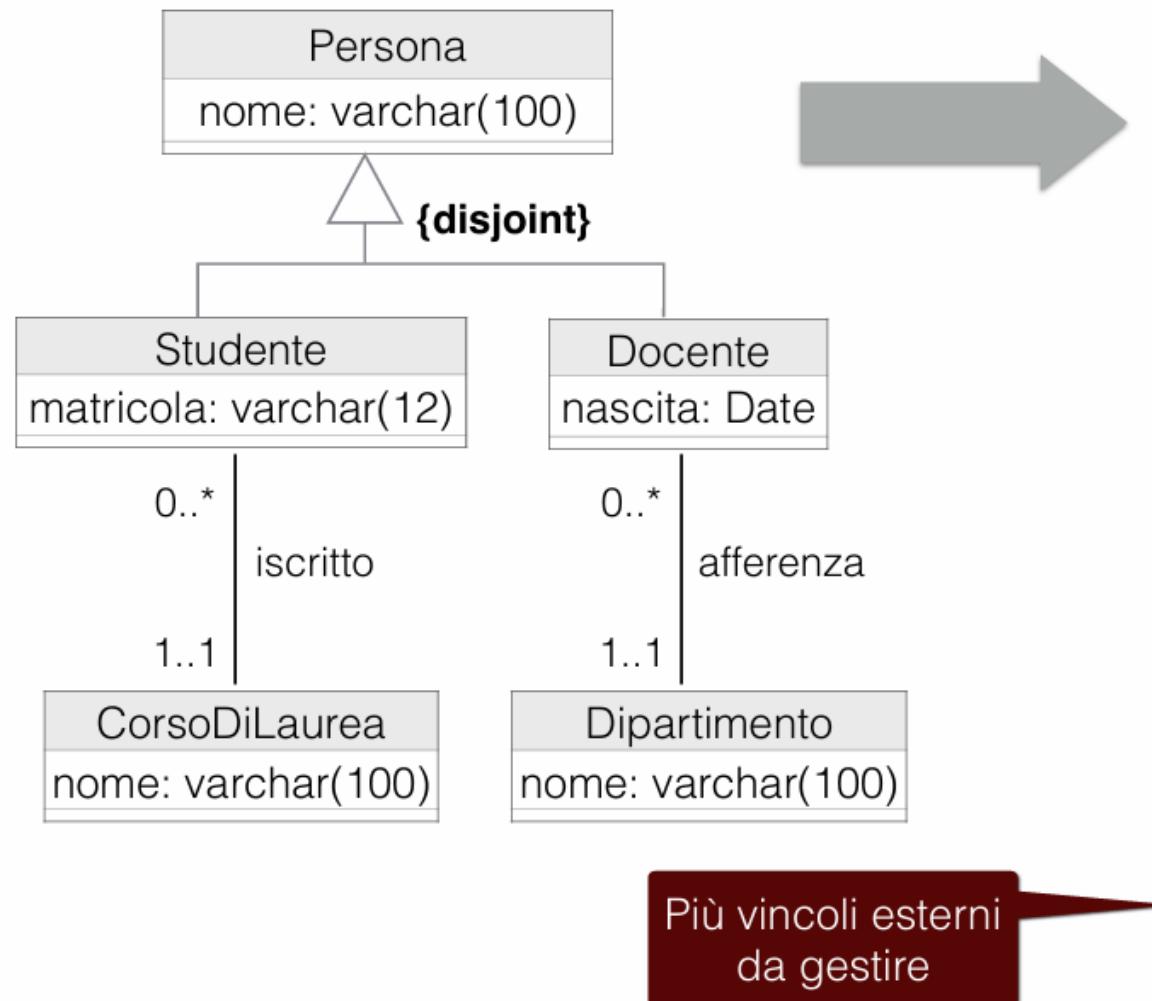
Approccio: fondere un intero livello della generalizzazione (superclasse + sottoclassi) in un'unica classe



Vincoli esterni: per ogni istanza p di Persona:

1. p.tipo = 'Studente' se e solo se p.matricola è valorizzato
2. p.tipo = 'Studente' se e solo se p è coinvolto in un link "iscritto"

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



```

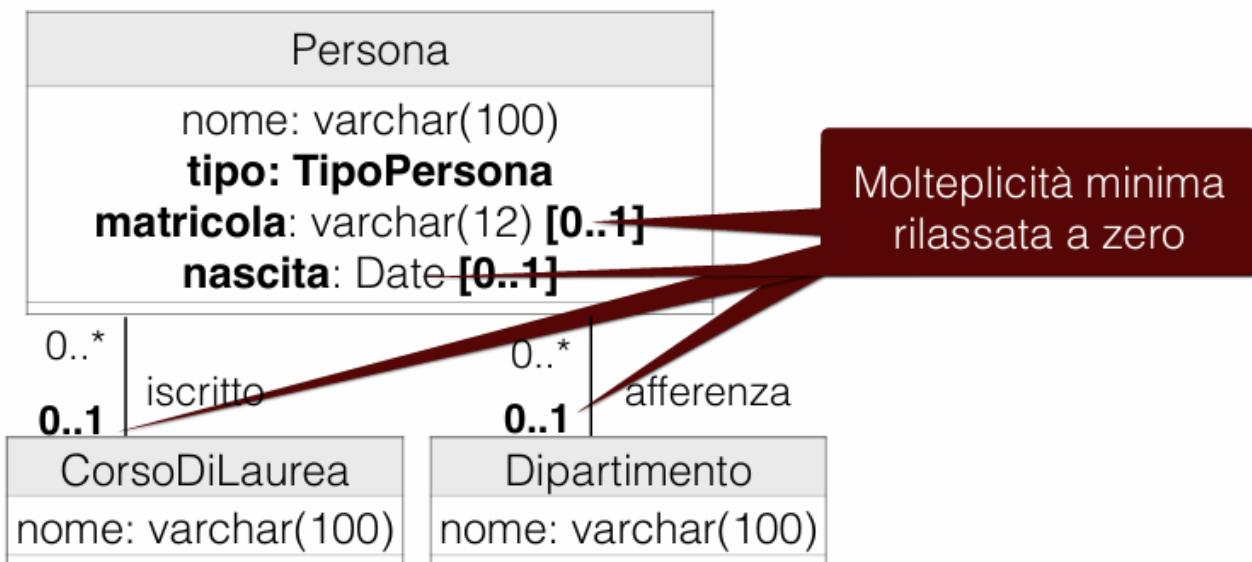
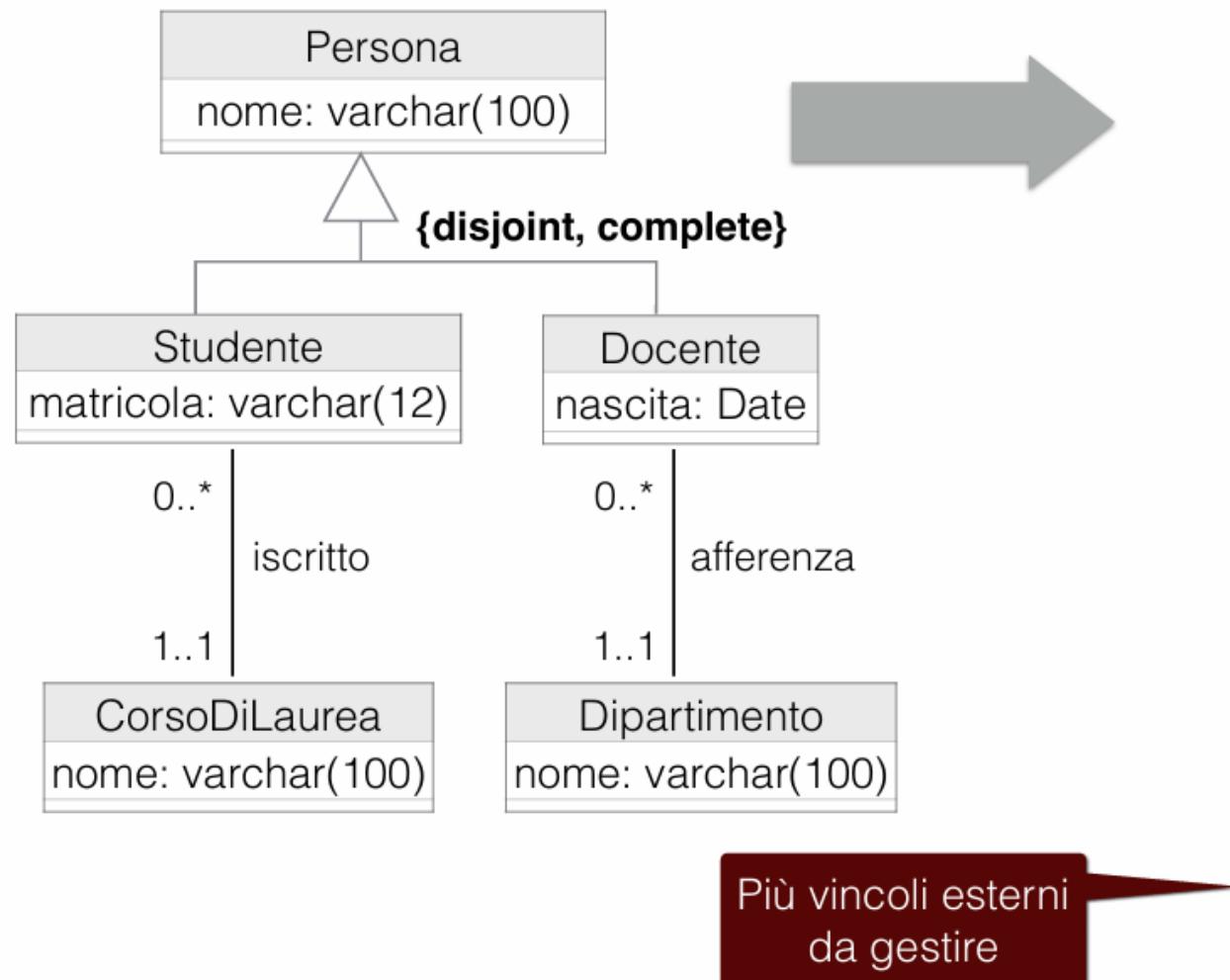
CREATE TYPE TipoPersona AS ENUM
('Persona', 'Studente', 'Docente');

```

Vincoli esterni: per ogni istanza p di Persona:

1. p.tipo = 'Studente' se e solo se p.matricola è valorizzato
2. p.tipo = 'Studente' se e solo se p è coinvolto in un link "iscritto"
3. p.tipo = 'Docente' se e solo se p.nascita è valorizzato
4. p.tipo = 'Docente' se e solo se p è coinvolto in un link "afferenza"

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



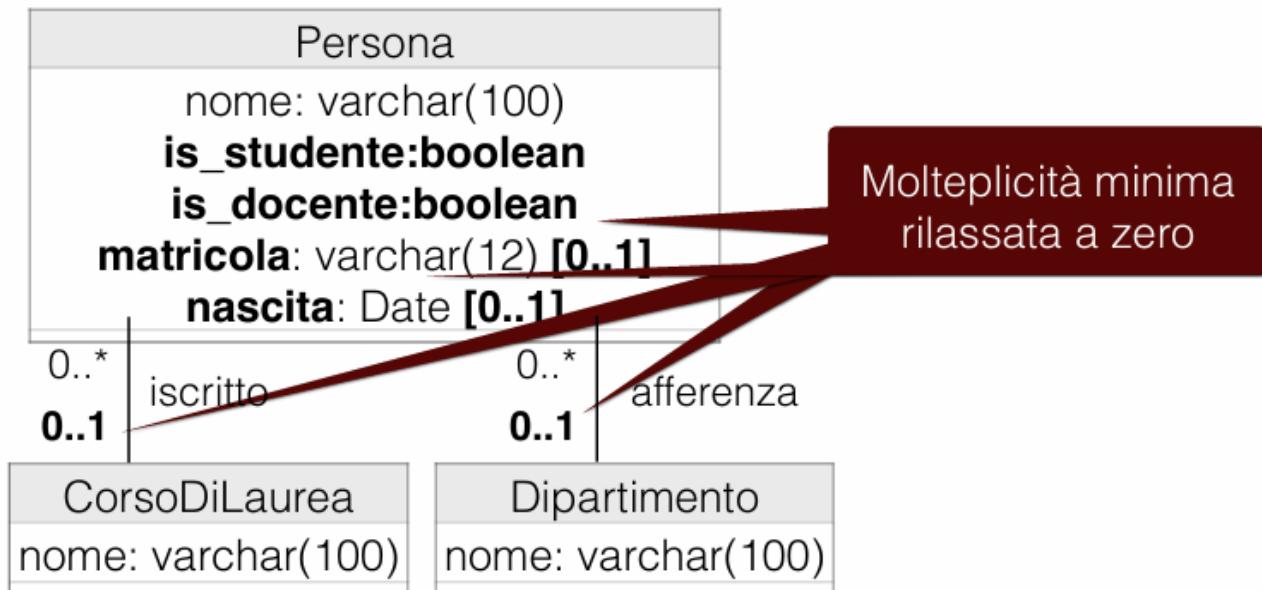
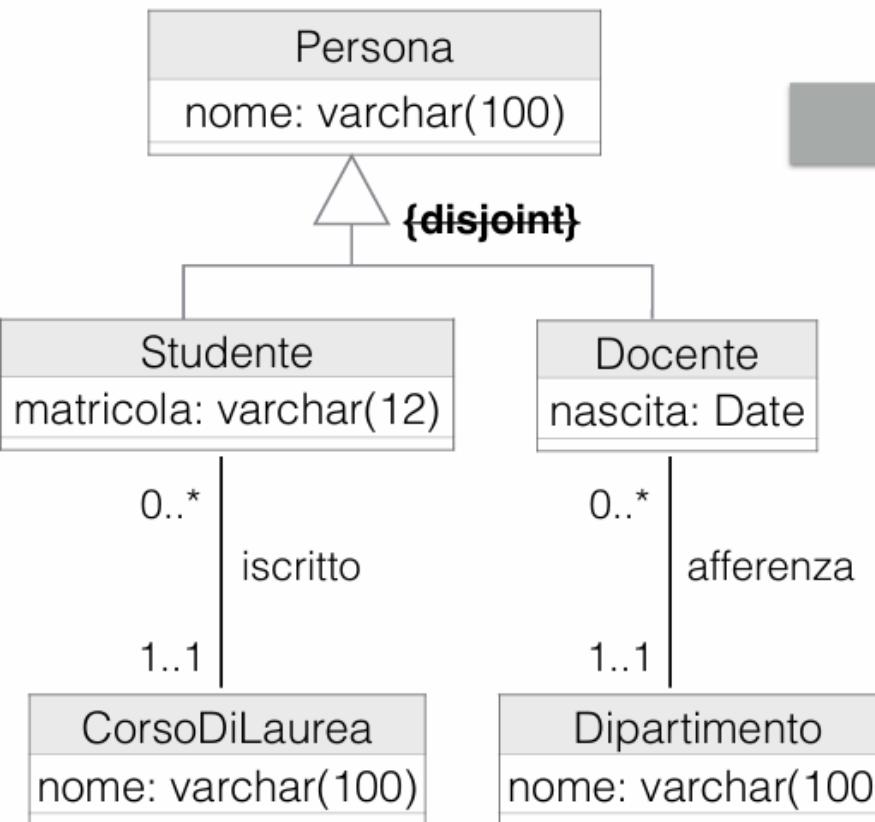
```

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
2. p.tipo = 'Studente' se e solo se p è coinvolto in un link "iscritto"
3. p.tipo = 'Docente' se e solo se p.nascita è valorizzato
4. p.tipo = 'Docente' se e solo se p è coinvolto in un link "afferenza"

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

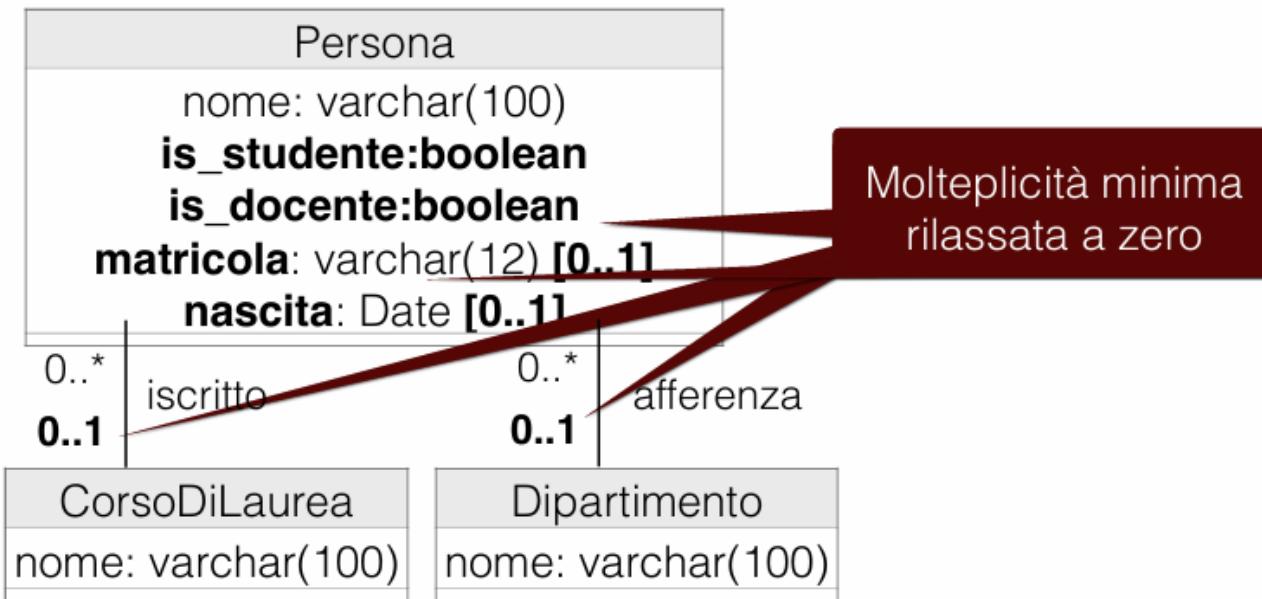
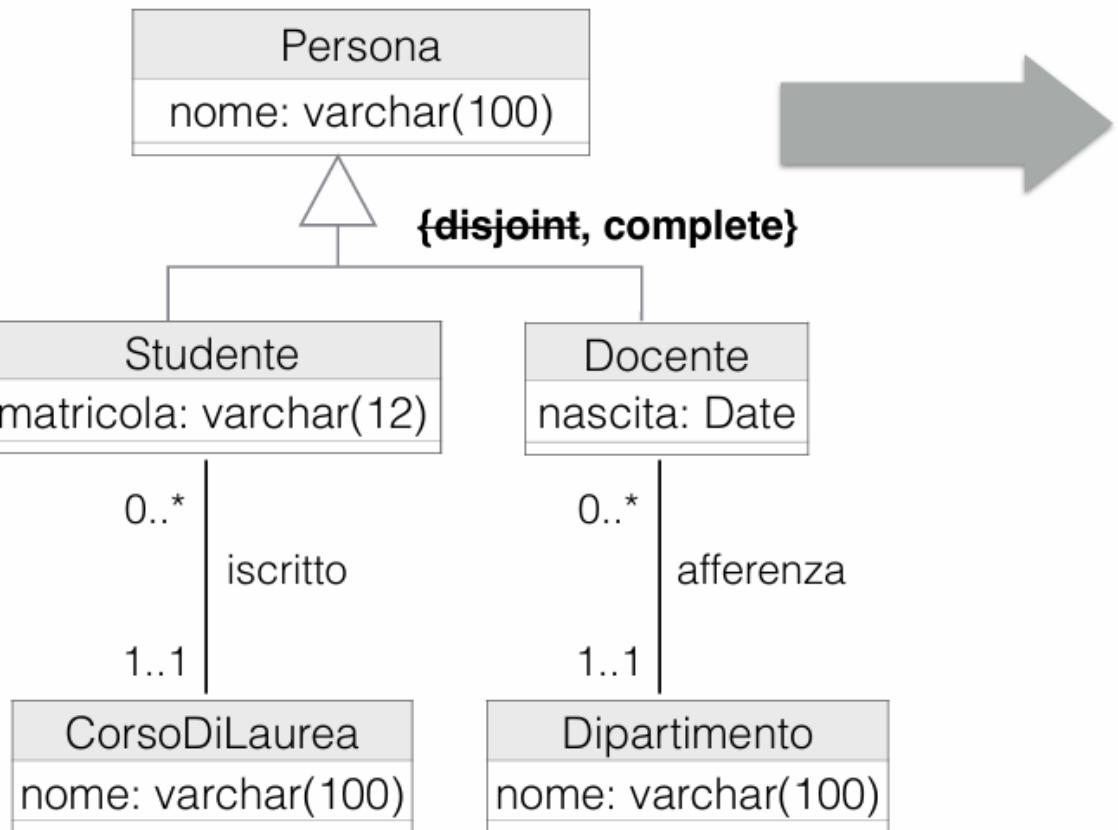


Molteplicità minima rilassata a zero

Vincoli esterni: per ogni istanza p di Persona:

1. $p.\text{is_studente} = \text{TRUE}$ se e solo se $p.\text{matricola}$ è valorizzato
2. $p.\text{is_studente} = \text{TRUE}$ se e solo se p è coinvolto in un link “iscritto”
3. $p.\text{is_docente} = \text{TRUE}$ se e solo se $p.\text{nascita}$ è valorizzato
4. $p.\text{is_docente} = \text{TRUE}$ se e solo se p è coinvolto in un link “afferenza”

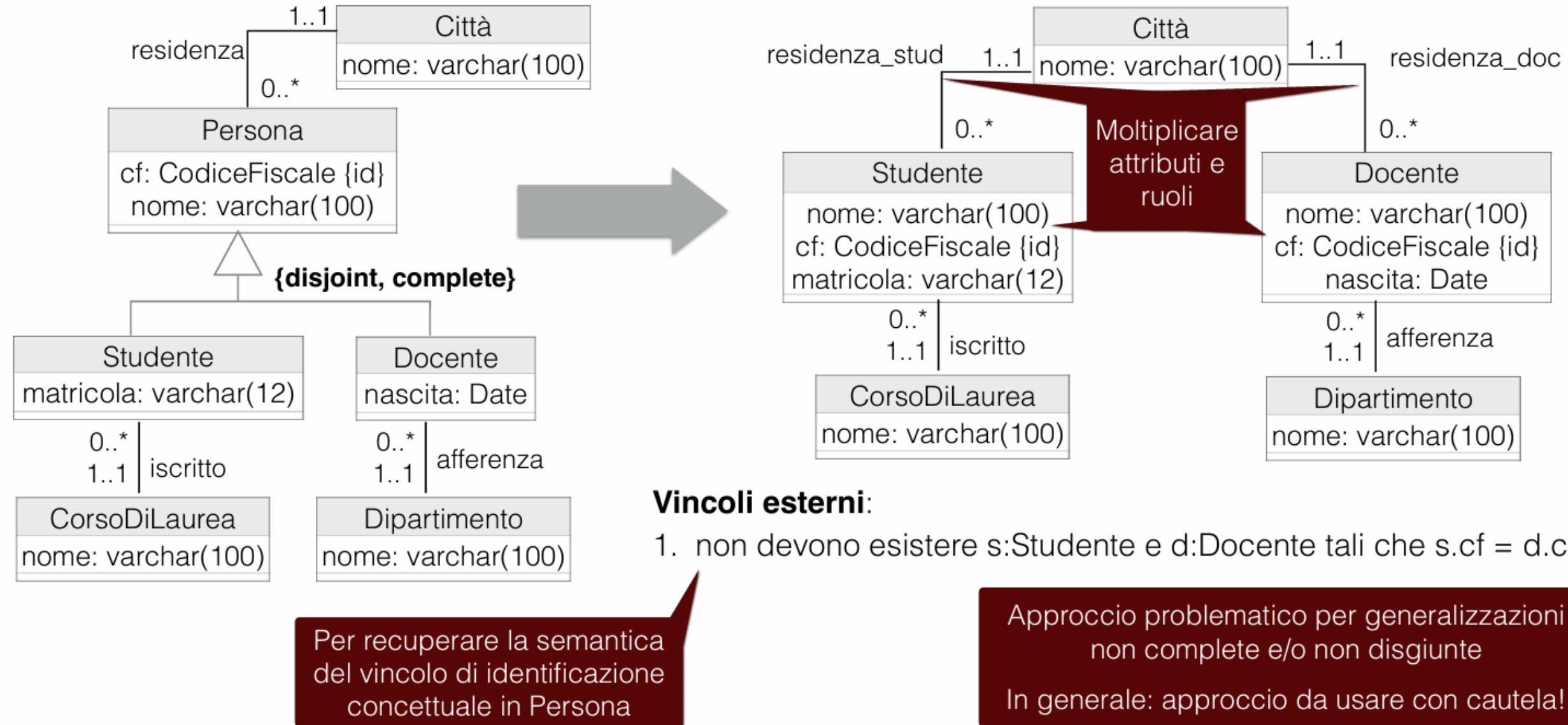
Ristrutturazione di generalizzazioni per fusione **in caso di sottoclassi disgiunte**



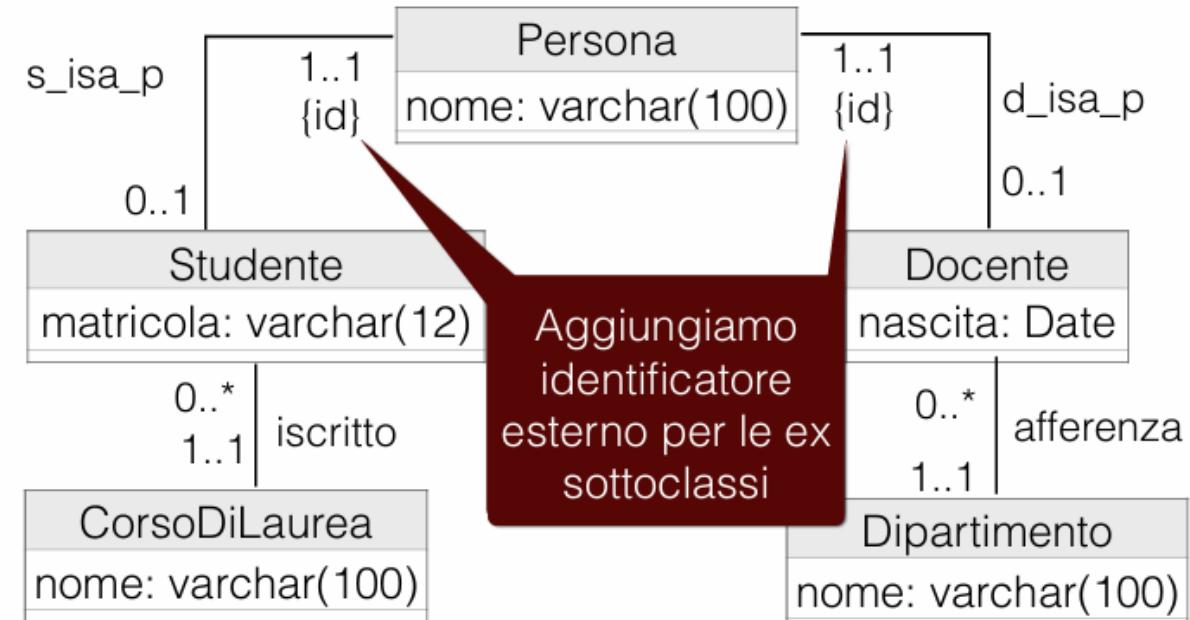
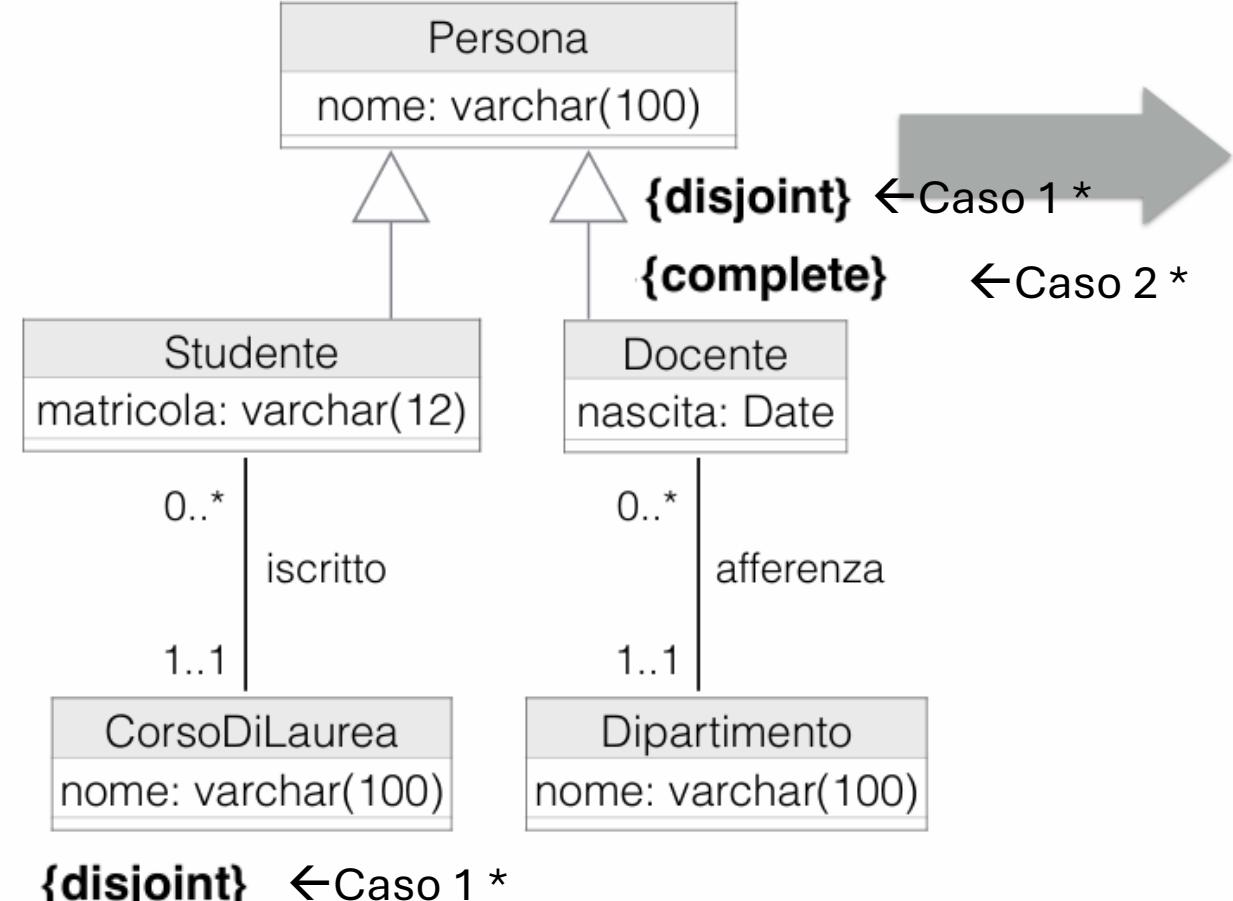
Vincoli esterni: per ogni istanza p di Persona:

1. p.is_studente = TRUE se e solo se p.matricola è valorizzato
2. p.is_studente = TRUE se e solo se p è coinvolto in un link "iscritto"
3. p.is_docente = TRUE se e solo se p.nascita è valorizzato
4. p.is_docente = TRUE se e solo se p è coinvolto in un link "afferenza"
5. **p.is_docente = TRUE oppure p.is_studente = TRUE**

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



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



Struttura perfettamente mantenuta

{complete} ←Caso 2 *

Vincoli esterni:

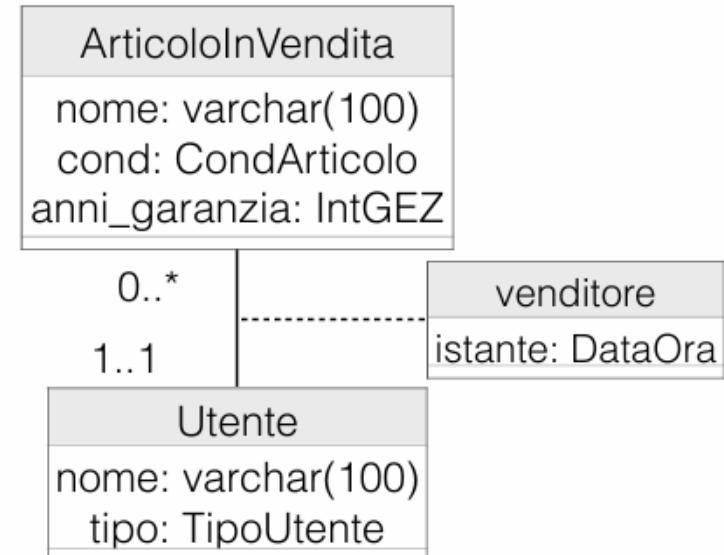
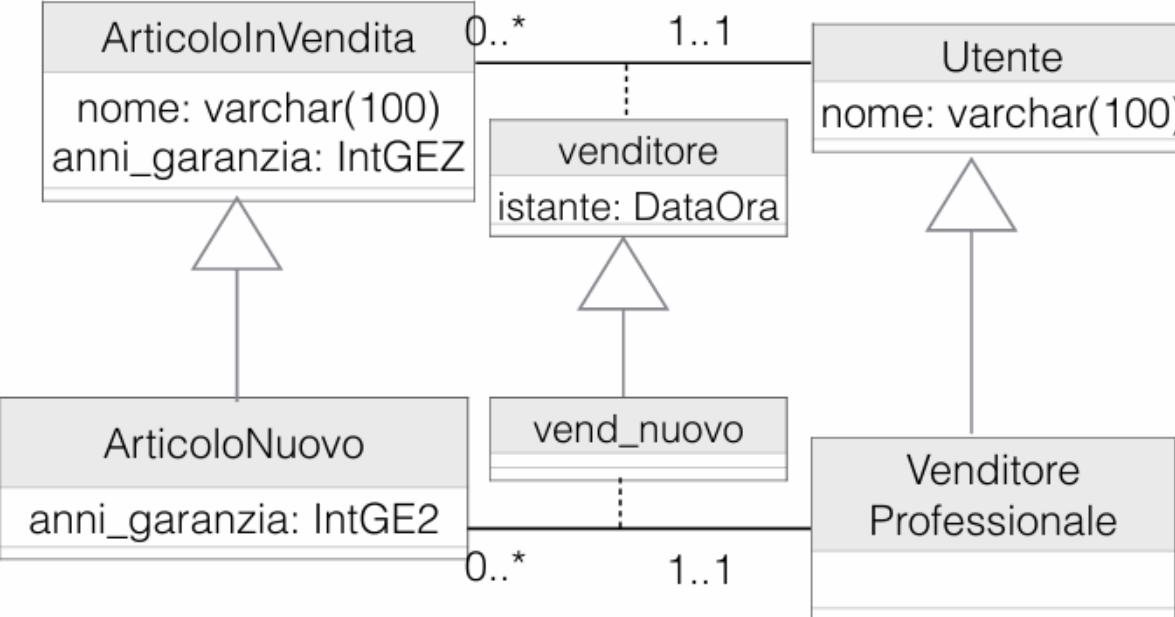
1. Disgiunzione: non devono esistere p:Persona, s:Studente e d:Docente tali che (s, p): s_isa_p e (d, p): d_isa_p
1. Completezza: per ogni p:Persona:
 - esiste s:Studente per cui: (s, p): s_isa_p
 - oppure esiste d:Docente per cui: (d, p): d_isa_p

{disjoint} ←Caso 1 *

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

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



```

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

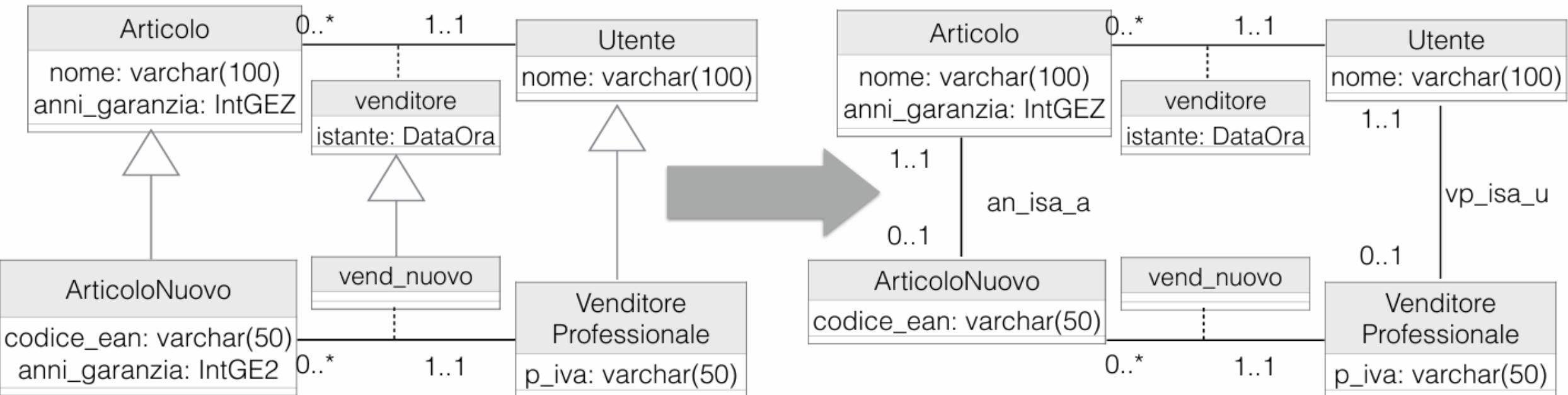
Vincoli esterni:

- per ogni a:ArticoloInVendita:
 - se a.cond = 'nuovo' allora a.anni_garanzia >= 2
 - se a.cond = 'nuovo' allora il link (a,u):venditore nel quale 'a' è coinvolto è tale che u.tipo = 'prof'

```

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

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



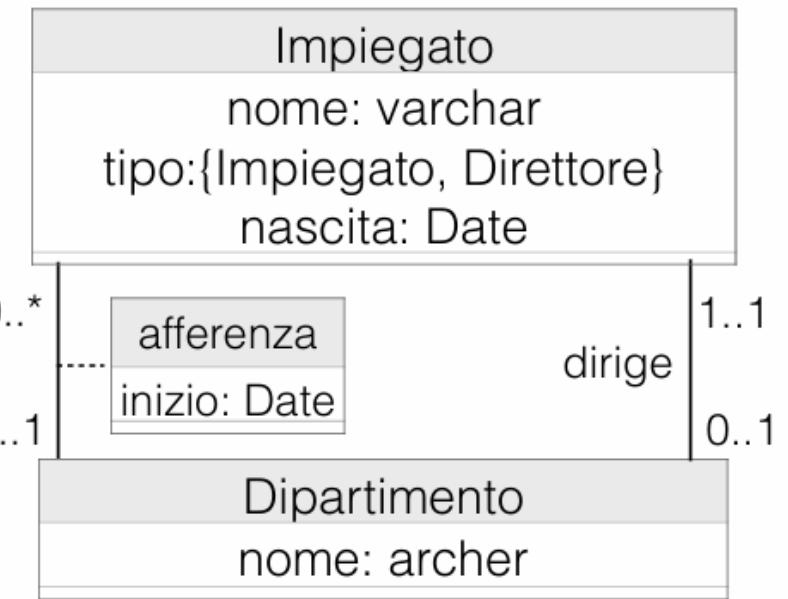
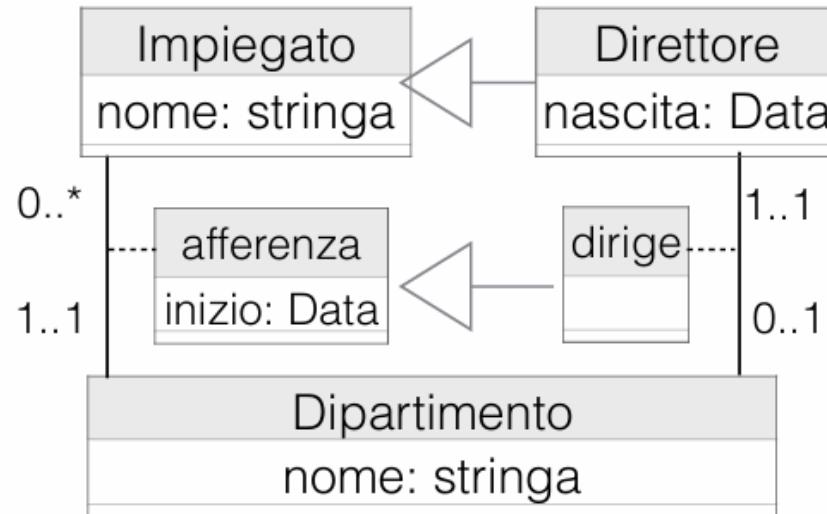
```

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

Vincoli esterni:

- per ogni **an:ArticoloNuovo**, l'istanza **a:Articolo** tale che **(an, a):an_isa_a** deve avere: **a.anni_garanzia >= 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**

Esempio:



Vincolo esterno:

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

[V.Dipartimento.direttore_anni_fferenza]

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

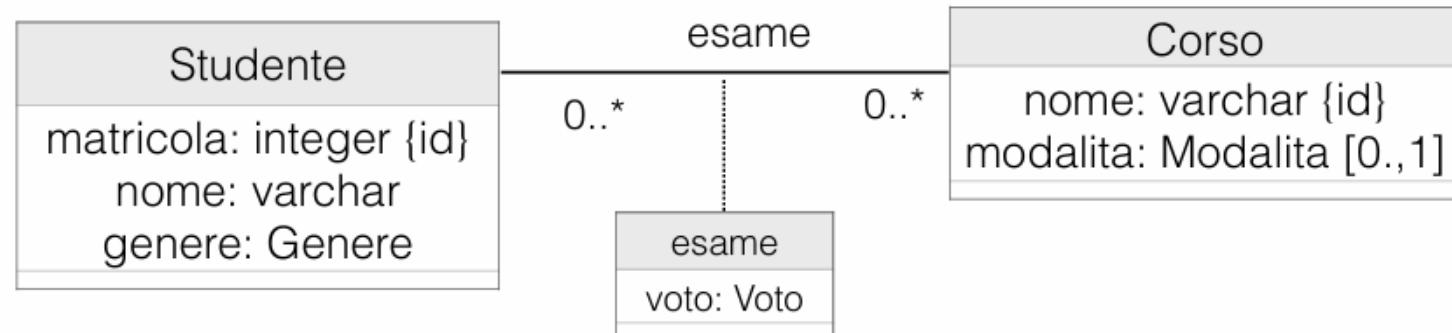
Vincolo esterno:

[V.Dipartimento.direttore_anni_fferenza]

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. $(dir, dip).inizio \leq adesso - 5 \text{ anni}$



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'

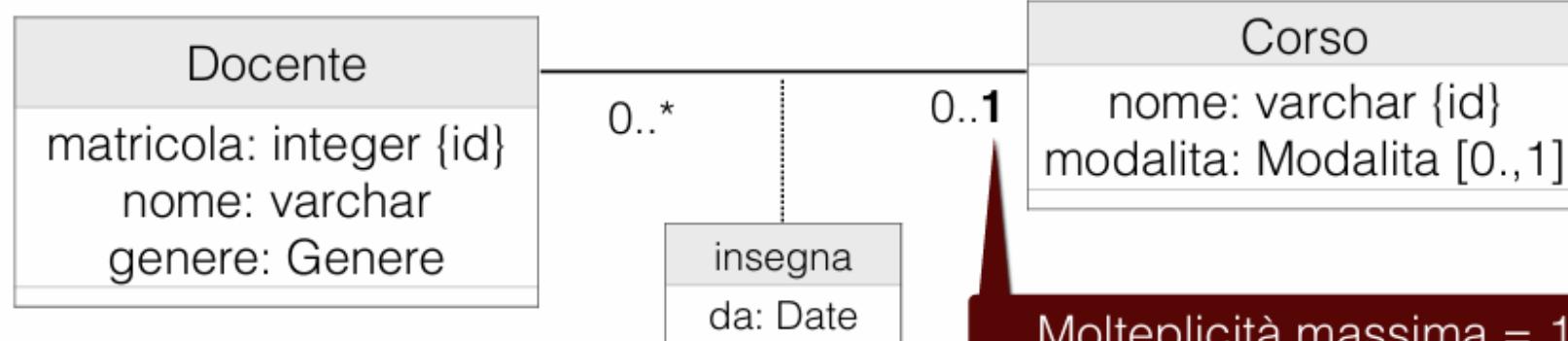
(*): può assumere valori NULL

```

CREATE TABLE esame (
    studente integer not null,
    corso integer not null,
    voto Voto not null,
    primary key (studente, corso),
    foreign key (studente) references
        Studente(matricola),
    foreign key (corso) references
        Corso(nome)
);

```

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 accorcia

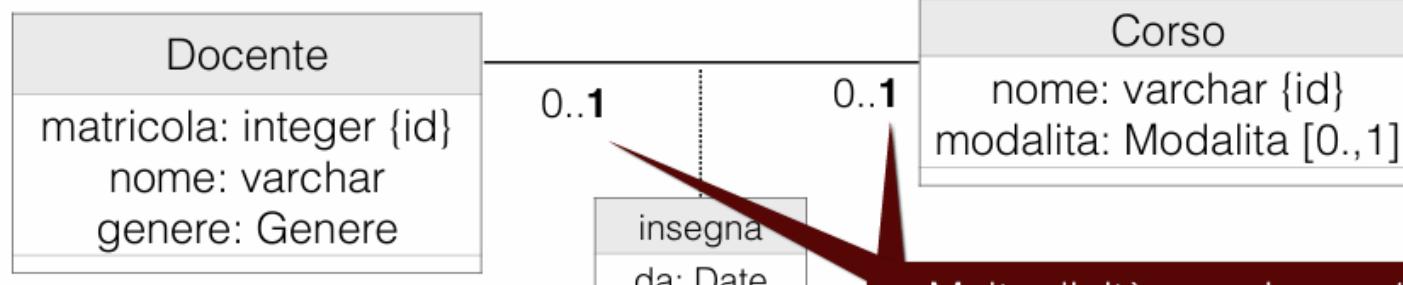
insegna (**docente:integer** , corso:varchar, da:Date)

~~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)

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

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

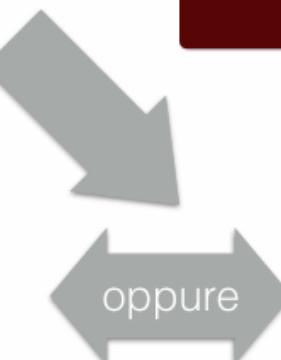
insegna (docente:integer, **corso:varchar**, da:Date)

~~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)

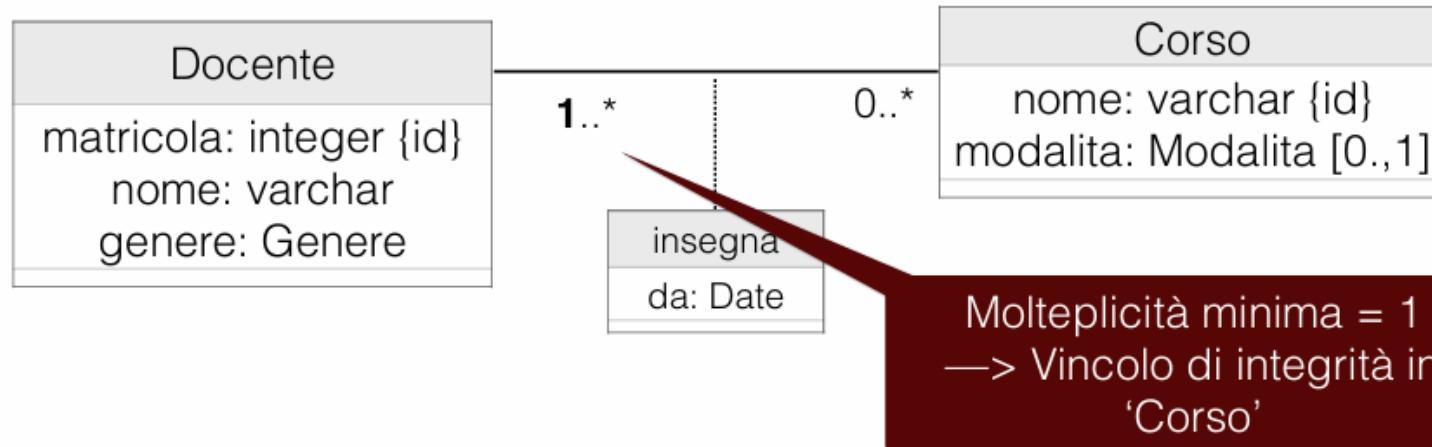
altra chiave: corso

~~altra chiave: docente~~

foreign key: docente references Docente(matricola)

foreign key: corso references Corso(nome)

Passo 2: Vincoli di molteplicità minima 1



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

Corso(nome:varchar, modalita*:Modalita)

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

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

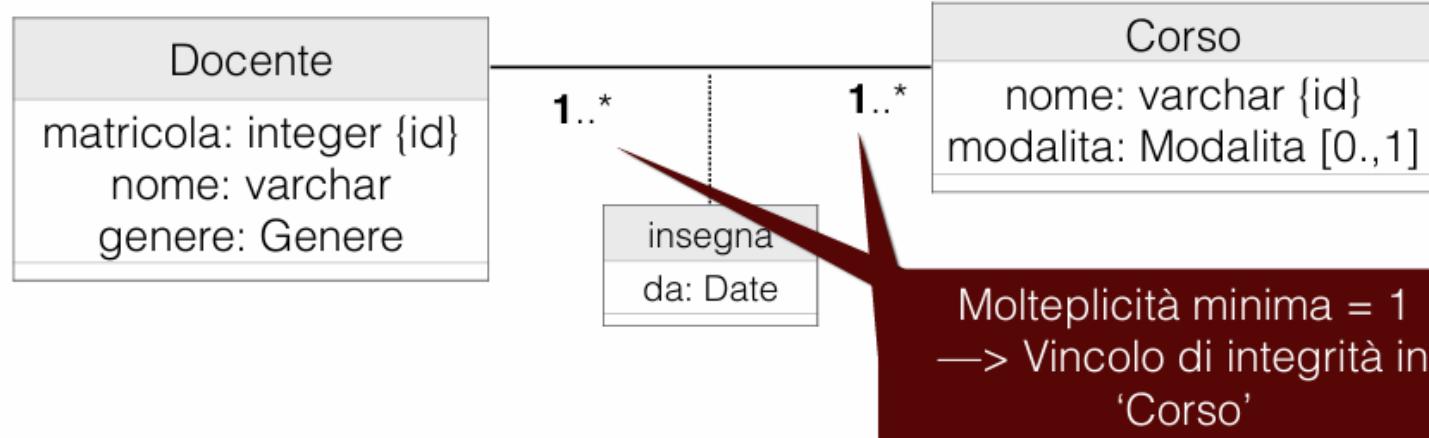
foreign key: docente references Docente(matricola)

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(coro)

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

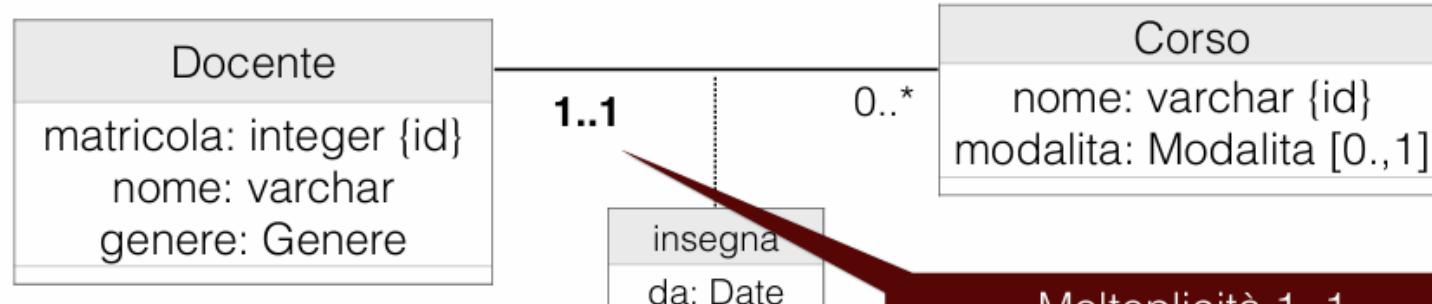
foreign key: docente references Docente(matricola)

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



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

Corso(nome:varchar, modalita*: Modalita)

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

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

foreign key: docente references Docente(matricola)

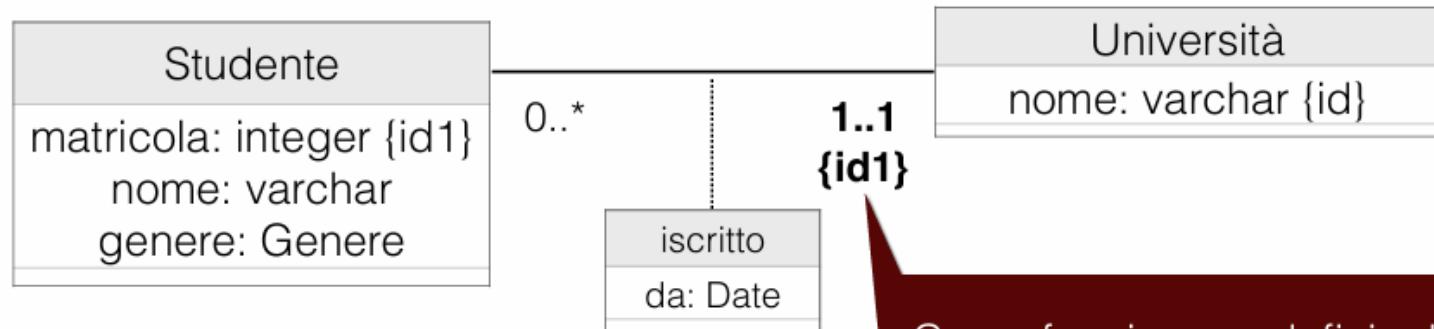
foreign key: corso references Corso(nome)

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(coro)

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?

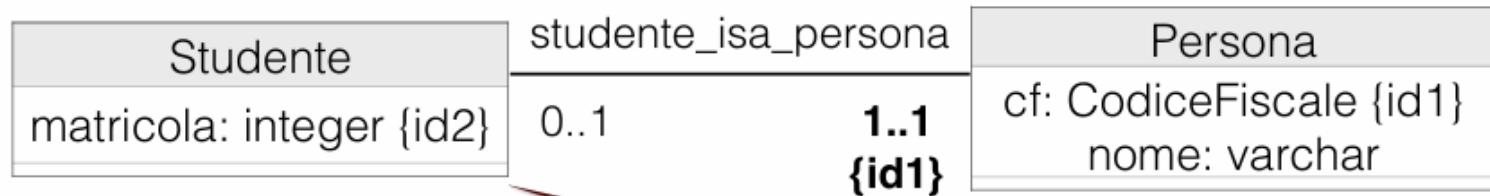
Universita(nome:varchar)

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?

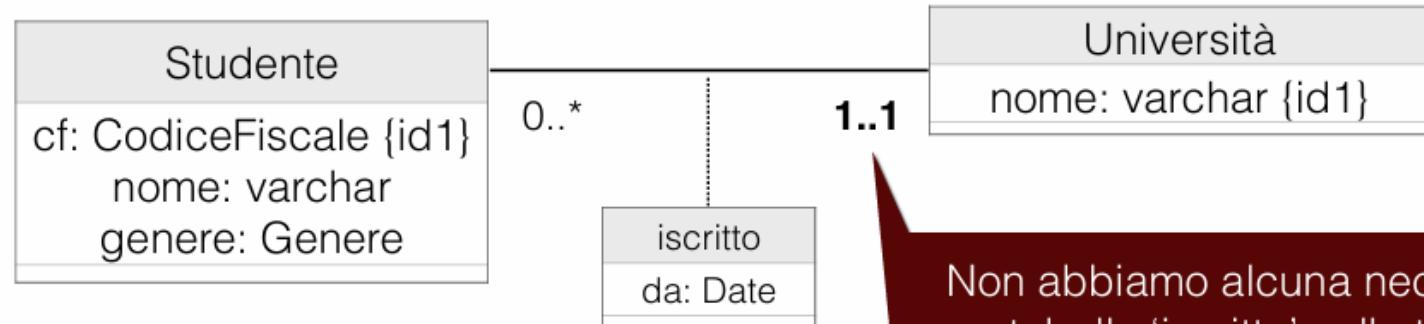
Persona(cf:CodiceFiscale, nome:varchar)

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



Senza accorpamento

Università
nome: varchar {id1}

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

Ma può essere conveniente.
Valutare caso per caso

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)

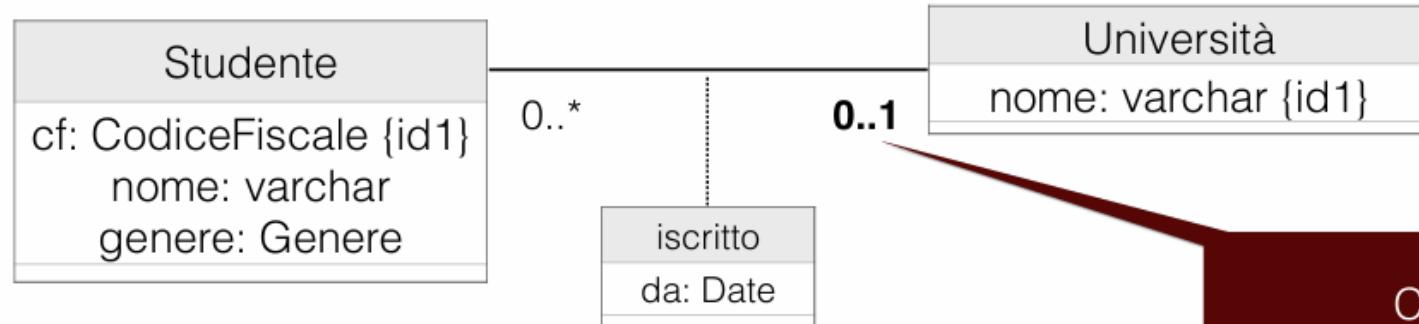
Con accorpamento

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)

✓. 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)

Con accorpamento

Universita(nome:varchar)

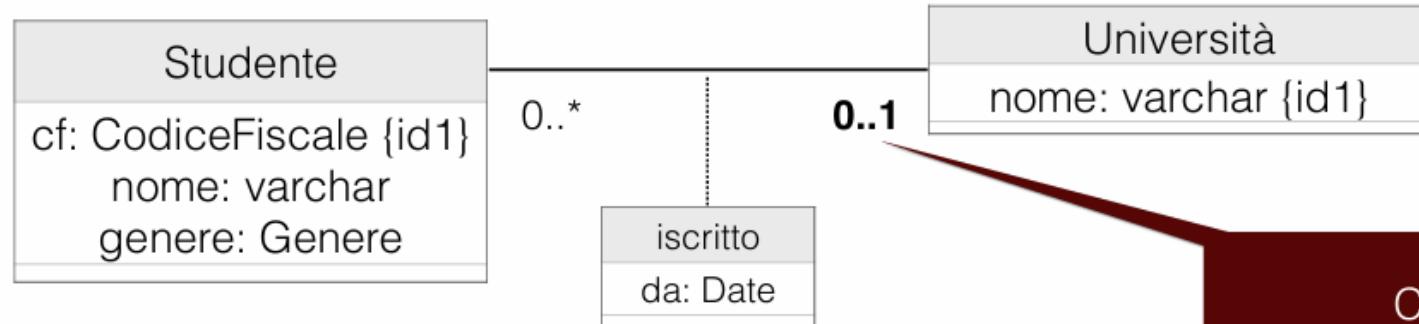
Studente(**cf:CodiceFiscale**, nome:varchar, genere:Genere,
universita*:varchar, **iscritto_da*:Date**)

foreign key: universita references Universita(nome)

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

(*) può assumere valori NULL

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)

✓. 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)

Con accorpamento

Universita(nome:varchar)

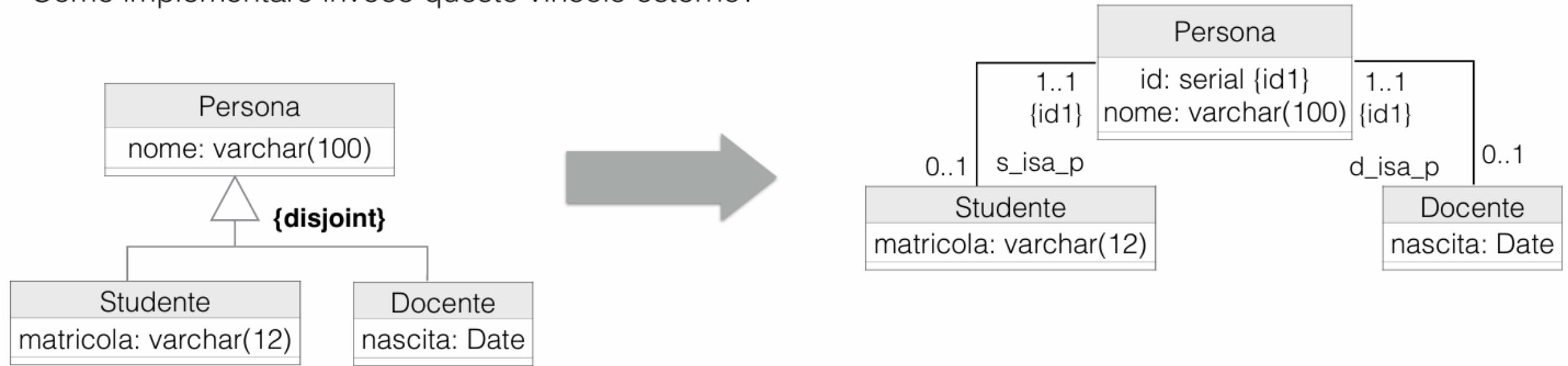
Studente(**cf:CodiceFiscale**, nome:varchar, genere:Genere,
universita*:varchar, **iscritto_da*:Date**)

foreign key: universita references Universita(nome)

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

(*) può assumere valori NULL

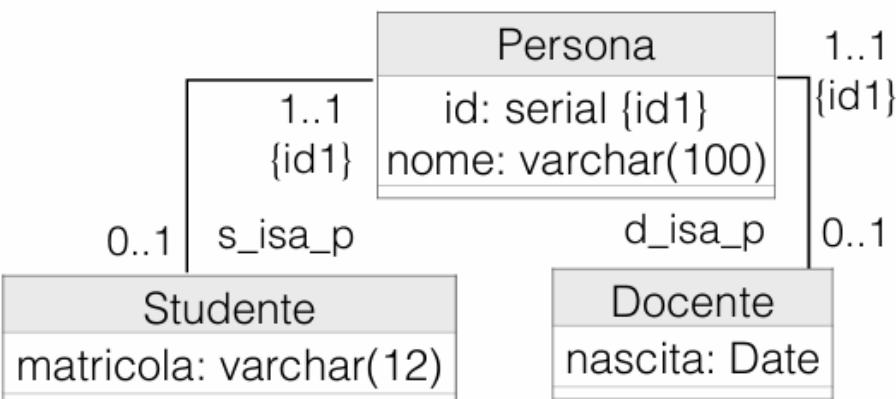
Come implementare invece questo vincolo esterno?



Vincolo esterno [V.Persona.gen.disj]

Non devono esistere $p: \text{Persona}$, $s: \text{Studente}$ e $d: \text{Docente}$ tali che $(s, p): s_{\text{isa}}_p$ e $(d, p): d_{\text{isa}}_p$

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



- 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

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

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

Vincolo esterno [V.Persona.gen.disj]

Non devono esistere p:Persona, s:Studente e d:Docente tali che (s, p): s_isap e (d, p): d_isap

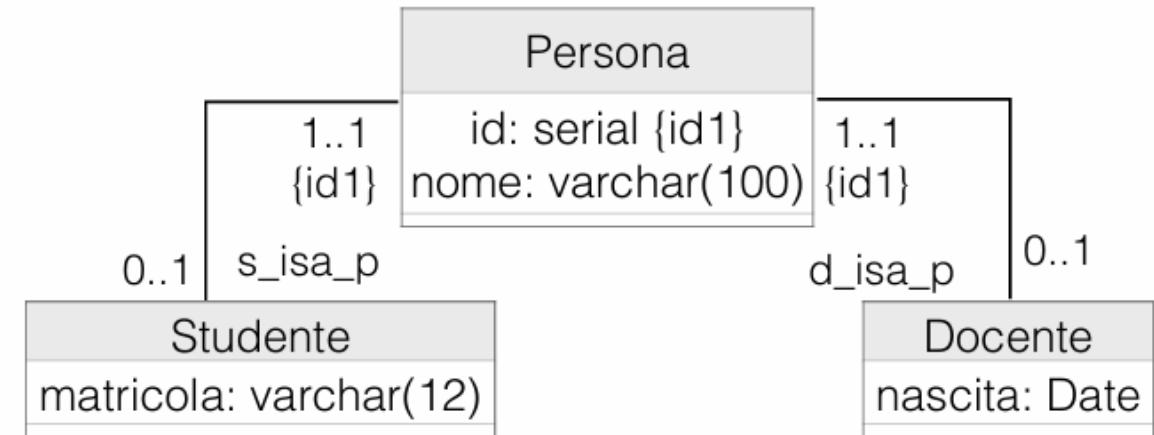
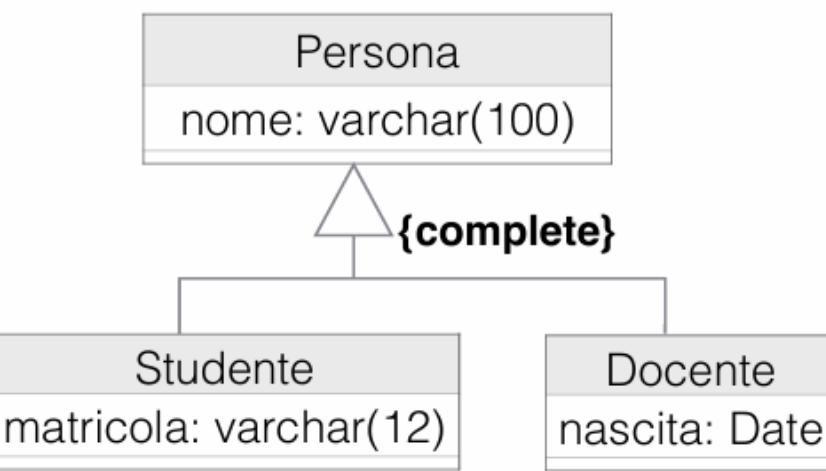
Quali sono gli eventi che possono portare un DB legale a diventare illegale?

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

Come implementare invece questo vincolo esterno?

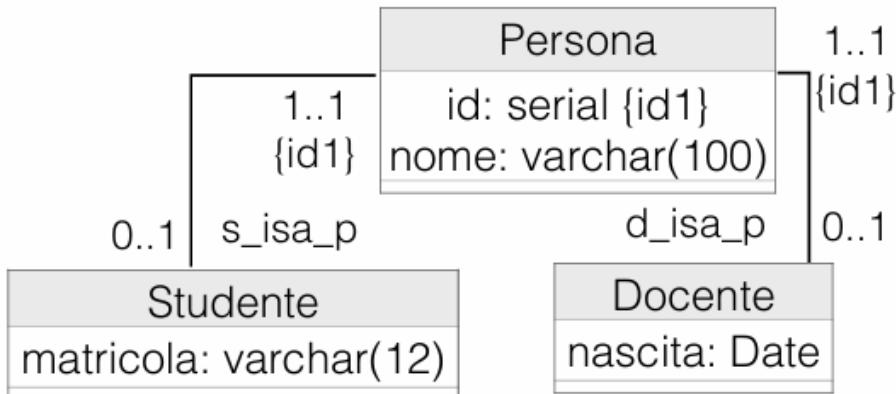


Vincolo esterno [V.Persona.gen.compl]

Per ogni $p: \text{Persona}$:

- esiste $s: \text{Studente}$ per cui $(s, p): s_{\text{isa}}_p$ oppure
- esiste $d: \text{Docente}$ per cui $(d, p): d_{\text{isa}}_p$

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]

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

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

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

Specifica use-case Studente

mediaVoti() : Reale in [18,31]

precondizioni: L'istanza this è coinvolta in almeno una istanza dell'associazione esame:

$$\exists c \text{ esame(this, } c).$$

postcondizioni:

Modifica del Livello Estensionale dei Dati: Nessuna

Valore di Ritorno: Detto $V = \{c, v \mid \text{esame(this, } c) \wedge \text{voto(this, } c, v)\}$,

$$\text{result} = \frac{\sum_{(c,v) \in V} v}{|V|}$$

Esempio:

```
CREATE OR REPLACE FUNCTION Studente_mediaVoti(_matr varchar)
    returns Reale_in_18_31 AS $BODY$
DECLARE
    matr varchar := NULL;
    media numeric := NULL;
BEGIN
    select s.matricola , avg(e.voto) INTO matr , media
    from Studente s LEFT OUTER JOIN esame e
        on s.matricola = e.studente
    where s.matricola = _matr
    group by s.matricola;

    IF (matr IS NULL) THEN
        RAISE EXCEPTION 'Error 001 – Lo studente non esiste';
    ELSE
        IF (media IS NULL) THEN
            RAISE EXCEPTION 'Error 002 – Lo studente non ha esami';
        ELSE
            RETURN media;
        END IF;
    END IF;
END;
$BODY$ LANGUAGE 'plpgsql';
```

Specifiche use-case Verbalizzazione

verbalizzaEsame(s : Studente, c : Corso, v : 18..31)

precondizioni: L'istanza s non è coinvolta in alcuna istanza dell'associazione esame con l'istanza c :

$$\neg \exists c \text{ esame}(s, c).$$

postcondizioni:

Modifica del Livello Estensionale dei Dati: Il livello estensionale dei dati al termine dell'esecuzione della funzione differisce da quello di partenza come segue:

Elementi del dominio di interpretazione : invariati

Variazioni nelle ennuple di predici: esame(s, c), voto(s, c, v).

Valore di Ritorno: nessuno.

Specifiche realizzativa use-case Verbalizzazione

verbalizzaEsame(s : varchar, c : integer, v : Voto)

algoritmo:

1. Esegui il seguente comando SQL:

insert into esame(studente, corso, voto) **values** (PAR_1, PAR_2, PAR_3)
dopo aver rimpiazzato 'PAR_1', 'PAR_2', 'PAR_3' con i valori dei parametri attuali,
rispettivamente, s , c , v .

2. Se il comando precedente restituisce un errore di vincolo di chiave violato, allora genera
l'errore 'Lo studente di matricola s ha già superato l'esame per il corso di codice c '.

Specifica use-case StatisticheStudenti

mediaVoti(s : Studente) : reale in [18,31]

precondizioni: L'istanza s è coinvolta in almeno un'istanza dell'associazione esame:
 $\exists c \text{ esame}(s, c)$.

postcondizioni:

Modifica del Livello Estensionale dei Dati: nessuna

Valore di Ritorno: result è tale da soddisfare la formula:

$\text{mediaVoti}_{\text{Studente}}(s, \text{result})$.

Specifica realizzativa use-case StatisticheStudenti

mediaVoti(s : varchar) : Reale_18_31

algoritmo:

1. Memorizza in result il risultato di:

SELECT Studente_mediaVoti(PAR_1) ;

dopo aver rimpiazzato 'PAR_1' con il valore del parametro attuale s .
(La query restituisce un reale o solleva una eccezione.)

2. Se la query ha lanciato una eccezione, rilancia l'errore e termina.
3. Altrimenti, restituisci result.

Specifica use-case StatisticheStudenti (continua)

numMedioEsami() : reale ≥ 0

precondizioni: Il livello estensionale dei dati definisce almeno una istanza di classe **Studente**:
 $\exists s \text{ Studente}(s)$.

postcondizioni:

Modifica del Livello Estensionale dei Dati: nessuna

Valore di Ritorno: **result** è pari al numero di istanze di associazione **esame** definite nel livello estensionale diviso per il numero di istanze di entità **Studente**. Formalmente, siano:

$$E = \{(s, c) \mid \text{esame}(s, c)\} \quad \text{e} \quad S = \{s \mid \text{Studente}(s)\}$$

gli insiemi, rispettivamente, di tutte le coppie (s, c) istanze dell'associazione **esame** e di tutte le istanze dell'entità **Studente**. Si ha: $\text{result} = \frac{|E|}{|S|}$.

Specifica realizzativa use-case StatisticheStudenti (continua)

numMedioEsami() : RealeNonNeg

algoritmo:

1. Esegui la seguente query SQL e memorizzane il risultato nella variabile *Q*:

```
select e.numEsami/s.numStudenti as numMedioEsami
from (select count(*) as numEsami from esame) e,
      (select count(*) as numStudenti from Studente) s
```

(La query restituisce un reale o un eccezione di ‘divisione per zero’.)

2. Se il risultato della query è un eccezione di ‘divisione per zero’, generare l’errore ‘Non esistono studenti’ e termina.
3. Altrimenti, restituisci *Q*.

End

Nota: Il dominio **RealeNonNeg** è stato scelto, all’inizio della Fase di Progettazione, come corrispondente del dominio concettuale **reale ≥ 0** .

Esempio:

(back-end web RESTful in Flask, <https://flask.palletsprojects.com>)

```
from flask import ...
from flask_restful import Resource ...
import psycopg2 # Driver Python per PostgreSQL
...

class MediaVoti(Resource):
    # Invocato dal front-end mediante chiamata HTTPS di tipo GET, ad es.:
    # GET https://app.site/statistichestudenti/mediavoti/<matricola>
    def get(self, matricola: str) -> "tuple[dict, int]":
        try:
            with get_db() as db:
                with db.cursor() as cur:
                    cur.execute(
                        "SELECT Studente_mediaVoti(%s) as media", (matricola,))
                    result = cur.fetchone() # leggi la prima tupla restituita
            return # crea oggetto JSON da restituire
            {
                "matricola": matricola,
                "media": result["media"]
            }, 200 # HTTP OK
        except psycopg2.DatabaseError as err:
            if (codice associato ad err == 001):

                abort(codice HTTP, description("Lo studente non esiste"))
            elif ...
        except Exception:
            abort(500, description="Internal Server Error")
```

```
from flask import ...
from flask_restful import Resource ...
import psycopg2 # Driver Python per PostgreSQL
...

class VerbalizzaEsame(Resource):
    # Invocato dal front-end mediante chiamata HTTPS di tipo POST, ad es.:
    # POST https://app.site/verbalizzazione/verbalizzaesame
    # con body JSON { studente:str , corso:int , voto:int }
    def post(self) -> "tuple[dict, int]":
        (studente, corso, voto) = leggi il body della richiesta POST
        try:
            with get_db() as db:
                with db.cursor() as cur:
                    cur.execute(
                        "INSERT INTO esame(studente, corso, voto) values (%s, %s, %s"
                        ),
                        (studente, corso, voto)
                    ) # puo' sollevare eccezioni
            ...
            return 201 # HTTP CREATED
        except psycopg2.DatabaseError as err:
            if (codice associato ad err == ...):
                abort(codice HTTP, description("Lo studente non esiste"))
            elif ...
        except Exception:
            abort(500, description="Internal Server Error")
...
```