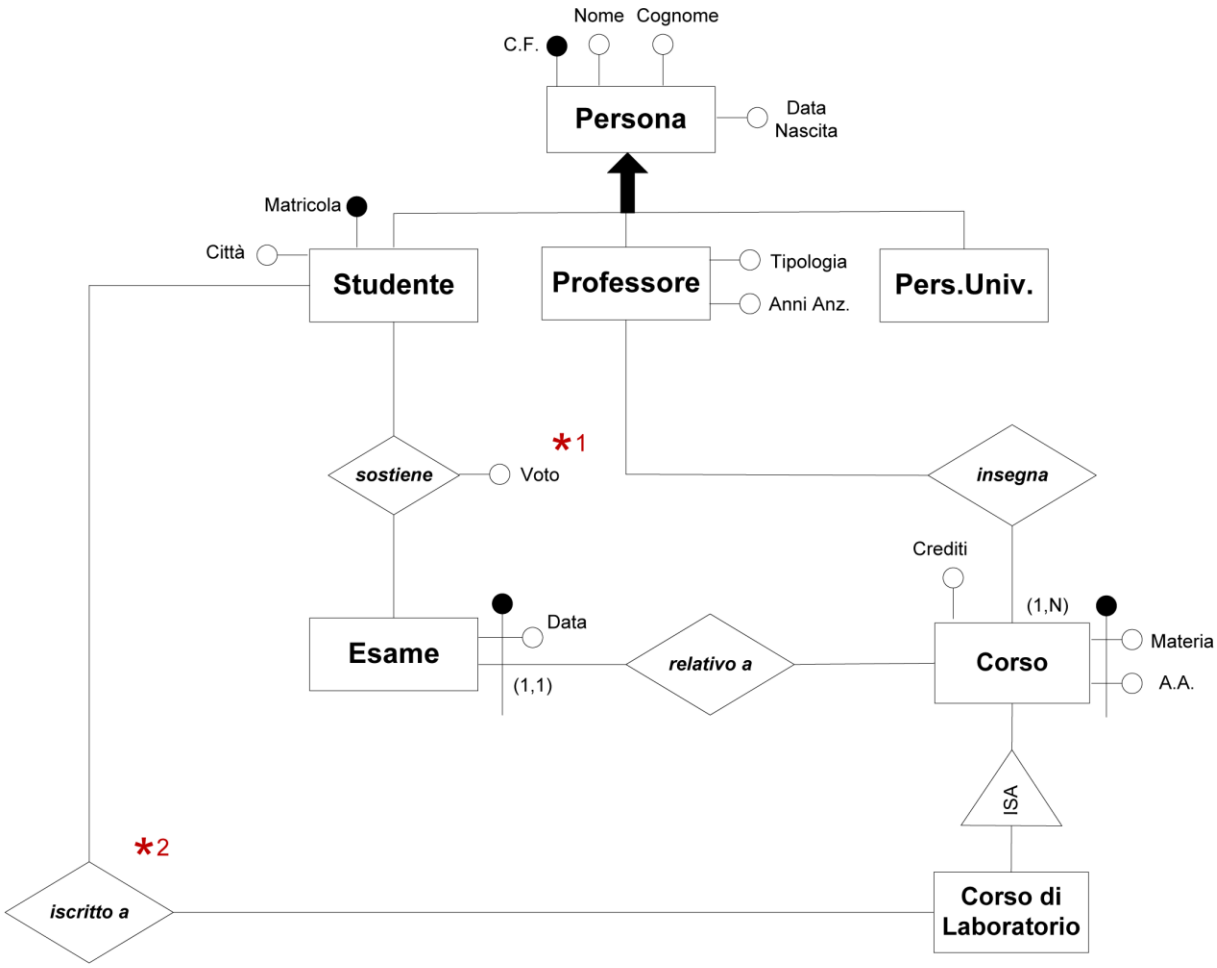
Esame di Basi di Dati

SOLUZIONE APPELLO 05/09/2011

1. Progettare lo schema Entità-Relazione di un'applicazione relativa ai corsi universitari organizzati dal Dipartimento di Informatica e Sistemistica. Di ogni corso interessa la materia di competenza, il numero di crediti, l'anno accademico in cui viene erogato e il/i Professore/i che lo eroga/no. Si noti che il numero di crediti di un corso può variare a seconda dell'anno accademico in cui è tenuto il corso stesso. Di ogni Professore interessa il codice fiscale (identificativo), il nome, il cognome, la data di nascita e la posizione occupata all'interno del Dipartimento (ad es., Ricercatore, Professore Associato, Professore Ordinario ecc.). Di ogni studente interessa il codice fiscale, il nome, il cognome, la data di nascita, la città di residenza e la matricola (identificativo). Il sistema tiene traccia anche del personale universitario (dottorandi, assegnisti di ricerca) che potrebbe in futuro divenire parte del gruppo dei professori; in questo caso sono di interesse codice fiscale (identificativo), nome, cognome e data di nascita. Per ogni corso, in uno specifico anno accademico vengono organizzati un certo numero di esami, identificati, oltre che dal corso a cui sono associati, anche dalla data prevista per l'esame. Ciascuno studente può sostenere più volte l'esame di un particolare corso, ed è rilevante conoscere il voto ottenuto (da 0 a 30 e Lode). La regolamentazione adottata dal Dipartimento prevede che per sostenere un esame di un corso non sia strettamente necessario aver seguito alcuna lezione del corso relativo. Esistono però particolari corsi – i corsi di laboratorio – in cui è necessario che ciascuno studente effettui un'iscrizione preventiva. In caso contrario, l'utente non potrà sostenere l'esame. A tale scopo, il sistema dovrà memorizzare l'elenco degli studenti iscritti ad ogni corso di laboratorio.

1.1 SCHEMA E-R

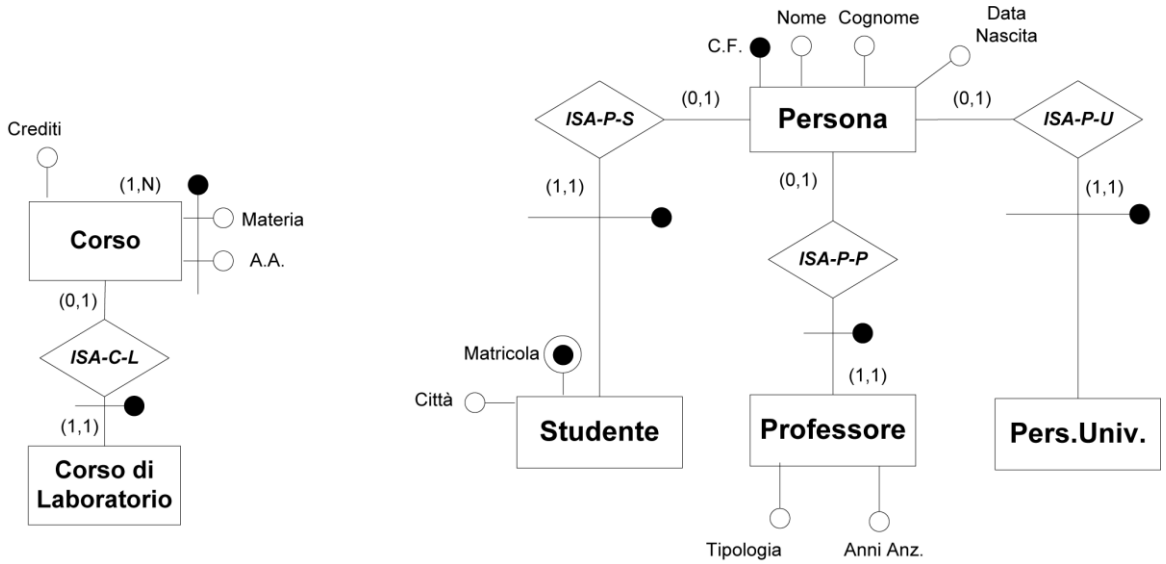


*1 L'attributo Voto può assumere solo valori che vanno da 0 a 30 e Lode

*2 Uno studente può sostenere un'esame relativo ad un particolare corso di laboratorio se e solo se è iscritto a quel corso

2. Si richiede di effettuare la progettazione logica dell'applicazione, producendo (in una qualunque notazione) lo schema relazionale completo di vincoli, seguendo l'indicazione di evitare valori nulli nella base di dati.

2.1 ESTRATTO DI SCHEMA E-R RISTRUTTURATO



Vincoli di Generalizzazione :

- Ogni istanza di **Persona** partecipa ad "**ISA-P-S**", oppure ad "**ISA-P-U**", oppure ad "**ISA-P-P**"

2.2 SCHEMA RELAZIONALE

Persona(CF,Nome,Cognome,DataNascita)

Studente(Matricola,CF,Città)
foreign key : Studente(CF) ⊆ Persona(CF)
chiave : CF

Professore(CF,Tipologia,AnniAnz)
foreign key : Professore(CF) ⊆ Persona(CF)

PersUniv(CF)
foreign key : PersUniv(CF) ⊆ Persona(CF)

Insegna(Professore,Corso,AACorso)
foreign key : Insegna(Corso,AACorso) ⊆ Corso(Materia,AnnoAccademico)
foreign key : Insegna(Professore) ⊆ Professore(CF)

Corso(Materia,AnnoAccademico,Crediti)
inclusione : Corso(Materia,AnnoAccademico) ⊆ Insegna(Corso,AACorso)

Esame(Materia,AACorso,Data)

foreign key : Esame(Materia,AACorso) ⊆ Corso(Materia,AnnoAccademico)

Sostiene(Studente,MateriaEsame,DataEsame,AACorso,Voto)
foreign key : Sostiene(Studente) ⊆ Studente(Matricola)
foreign key : Sostiene(MateriaEsame,DataEsame,AACorso) ⊆ Esame(Materia,Data,AACorso)
vincolo esterno : 0 <= Sostiene(Voto) <= 30 e Lode
vincolo esterno : ***if*** Sostiene(MateriaEsame[x]) = CorsoDiLab (Materia[x1])
then Sostiene(MateriaEsame[x],Studente[y]) ⊆ Iscritto(Studente,Materia)

CorsoDiLab (Materia,AnnoAccademico)
foreign key : CorsoDiLab(Materia,AnnoAccademico) ⊆ Corso(Materia,AnnoAccademico)

Iscritto(Studente,Materia,AACorsoLab)
foreign key : Iscritto(Studente) ⊆ Studente(Matricola)
foreign key : Iscritto(Materia,AACorsoLab) ⊆ CorsoLab(Materia,AnnoAccademico)

I vincoli di generalizzazione dello schema ristrutturato devono essere espressi in forma insiemistica nello schema relazionale :

- Studente [CF] ∩ PersUniv [CF] ∩ Professore [CF] = ∅
- Persona [CF] = Studente [CF] ∪ PersUniv [CF] ∪ Professore [CF]

3. Si consideri uno schema relazionale in cui la relazione *Sciatori(ID,Nome,Cognome,Età)* memorizza i vari sciatori di una stagione sciistica, con il codice identificativo, il nome, il cognome e l'età. La relazione *Iscritto(SID, CodiceCorso, Settimana)* specifica il codice dei singoli corsi a cui i vari sciatori hanno partecipato, e la settimana (rappresentata attraverso un numero intero) in cui si è tenuto il corso. Si chiede di esprimere in SQL le seguenti interrogazioni:

3.1 – Individuare l'età più comune tra gli sciatori che hanno preso parte ad almeno un corso

```
CREATE VIEW EtàComune(Età,numEtà) AS
SELECT Età, count(Età)
FROM Sciatori, Iscritto
WHERE id=sid
GROUP BY Età

SELECT Età
FROM EtàComune
WHERE numEtà = (SELECT max(numEtà) FROM EtàComune)
```

3.2 – Trovare nome e cognome degli sciatori che hanno sciato almeno una settimana con un parente (i parenti sono quegli sciatori con lo stesso cognome).

```
SELECT distinct S1.Nome, S1.Cognome
FROM Sciatori S1, Iscritto I1
WHERE S1.Id = I1.Sid AND EXISTS (SELECT *
                    FROM  Iscritto I2, Sciatori S2
                    WHERE I2.sid <> I1.sid AND
                        I1.settimana=I2.settimana AND
                        I2.sid = S2.Id AND
                        S1.Cognome = S2.Cognome)
```

4. Lo schema relazionale ottenuto al punto 2 contiene alcuni vincoli esterni e vincoli di generalizzazione. Si richiede di scegliere due vincoli (a discrezione dello studente) e di tradurli in SQL, utilizzando i costrutti appropriati (ad es., vincoli di check/asserzioni SQL).

Esempio : traduzione in SQL dei vincoli di generalizzazione

- Studente [CF] ∩ PersUniv [CF] ∩ Professore [CF] = ∅

```
CREATE ASSERTION Studente_DISJOINT_Professore_DISJOINT_PersUniv(
check ( NOT EXISTS (
            SELECT CF
            FROM Studente
            INTERSECT
            SELECT CF
            FROM Professore )
      AND NOT EXISTS (
            SELECT CF
            FROM Studente
            INTERSECT
            SELECT CF
            FROM PersUniv )
      AND NOT EXISTS (
            SELECT CF
            FROM Professore
            INTERSECT
            SELECT CF
            FROM PersUniv )
      )
```

- Persona [CF] = Studente [CF] ∪ PersUniv [CF] ∪ Professore [CF]

```
CREATE ASSERTION Studente_PersUniv_Professore_COMPLETE_Persona (
check (
      (1 = SELECT count(*)
      FROM Studente, Persona
      WHERE Studente.CF = Persona.CF )
      OR
      (1 = SELECT count(*)
      FROM Professore, Persona
      WHERE Professore.CF = Persona.CF )
      OR
      (1 = SELECT count(*)
      FROM PersUniv, Persona
      WHERE PersUniv.CF = Persona.CF )
      )
```