Basi di dati

Appello del 04-12-2006

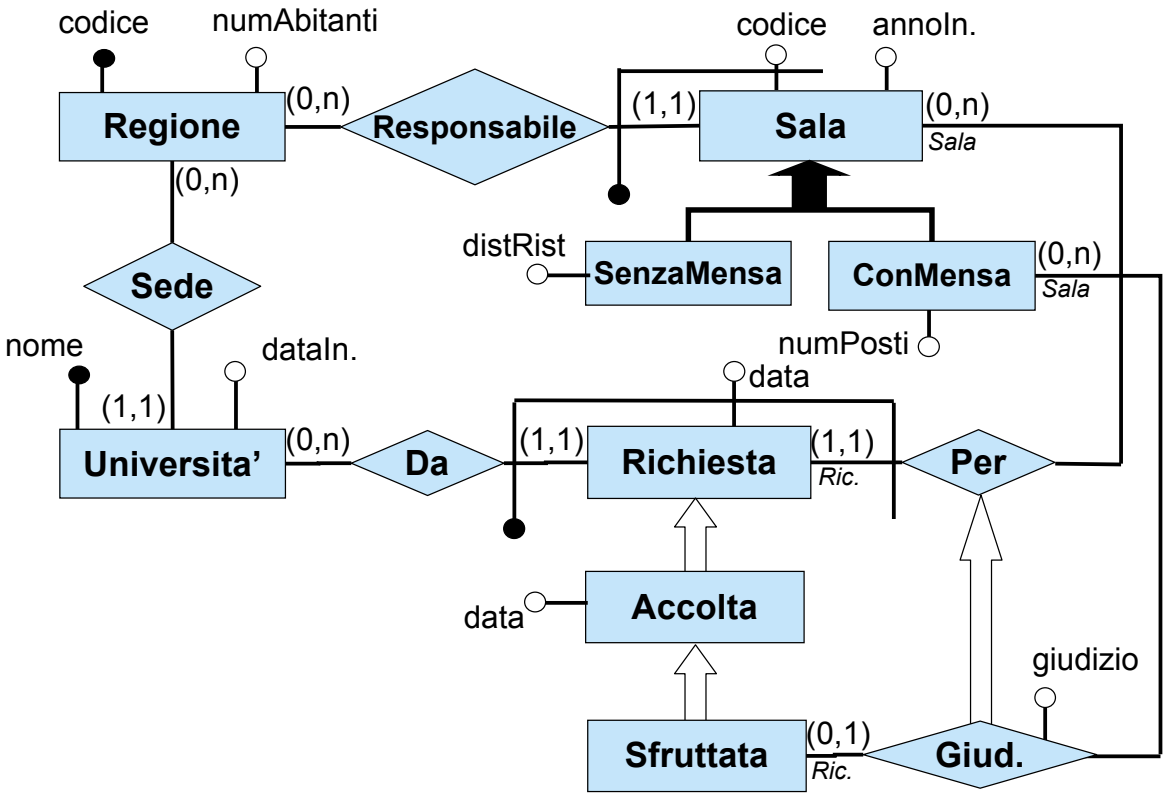
Soluzione Compito B

Anno Accademico 2006/07

Problema 1

Si richiede di progettare lo **schema concettuale Entità-Relazione** di un'applicazione relativa alla gestione delle sale congresso date in uso dalle regioni alle università. Di ogni regione interessa il codice (identificativo) ed il numero di abitanti. Si noti che esistono regioni di interesse per la nostra applicazione che attualmente non hanno sale congresso in gestione. Di ogni sala congresso interessa la regione responsabile della gestione, il codice (che identifica la sala congresso nell'ambito della regione responsabile) e l'anno di inaugurazione. Esistono esattamente due tipi di sale congresso: con mensa e senza mensa. Delle prime interessa il numero di posti della mensa, e delle seconde interessa la distanza dal ristorante più vicino. Di ogni università interessa il nome (identificativo), la data di inaugurazione, e la regione in cui ha sede. All'applicazione interessano le richieste di uso di sale congresso presentate dalle università. Ogni richiesta è presentata da esattamente un'università e riguarda l'uso di esattamente una sala congresso in una certa data. Alcune delle richieste sono accolte dalla regione, e di esse interessa la data di accoglimento. Solo per le richieste che riguardano le sale congresso con mense, e che sono state accolte e poi sfruttate dall'università (cioè a fronte delle quali l'università ha poi effettivamente utilizzato la sala congresso), l'università stessa ha la possibilità (non l'obbligo) di esprimere un giudizio sulla qualità della mensa, e tale giudizio è di interesse per l'applicazione.

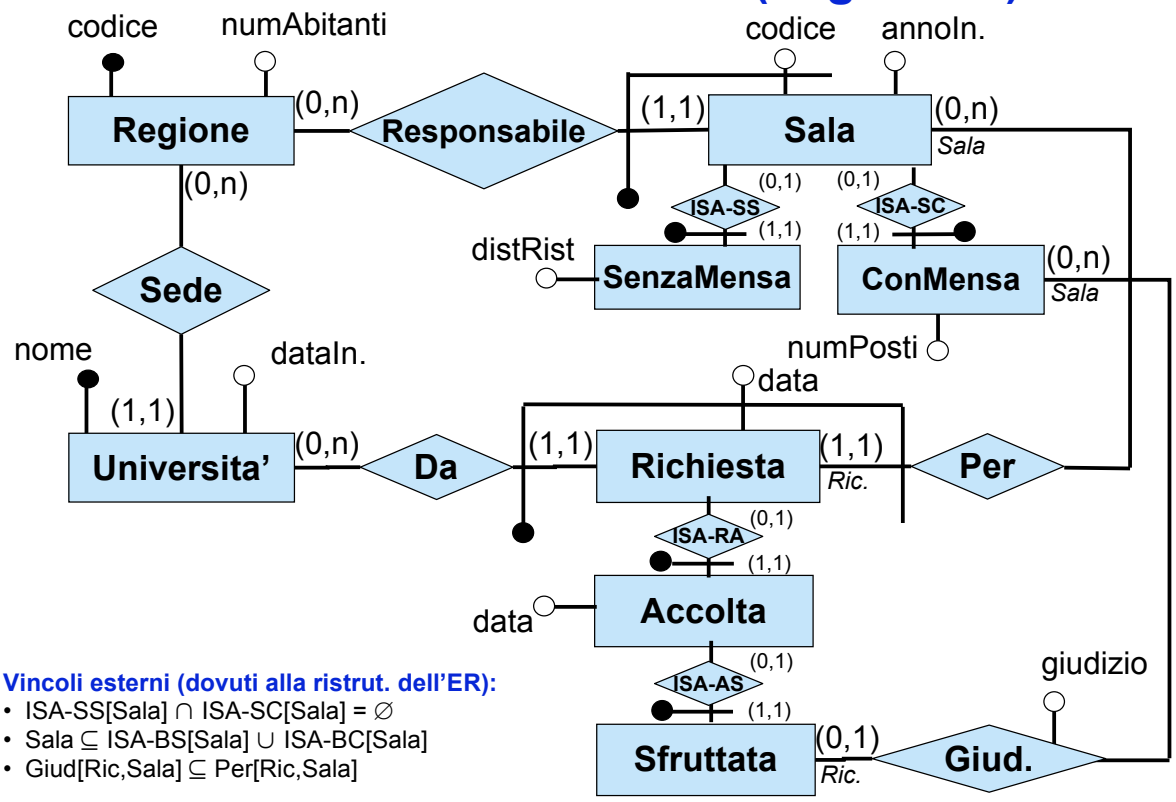
Schema ER



Problema 2

Si richiede di effettuare la **progettazione logica** dell'applicazione, producendo (in qualunque forma) lo schema relazionale completo di vincoli, seguendo l'indicazione che quando si accede ai dati relativi ad una università si vuole spesso conoscere la regione in cui ha sede.

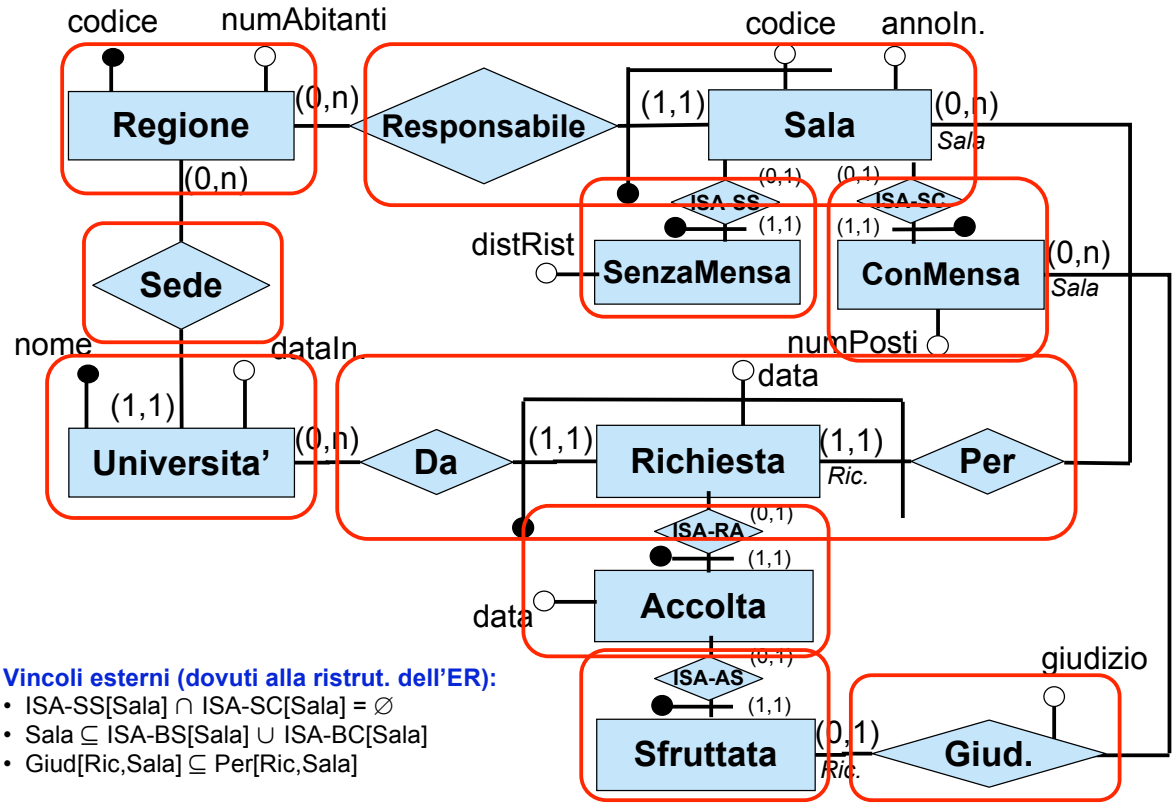
Schema ER ristrutturato (degradato)



Vincoli esterni (dovuti alla ristrut. dell'ER):

- ISA-SS[Sala] ∩ ISA-SC[Sala] = ∅
- Sala ⊆ ISA-BS[Sala] ∪ ISA-BC[Sala]
- Giud[Ric,Sala] ⊆ Per[Ric,Sala]

Schema logico: traduzione diretta del ER



Vincoli esterni (dovuti alla ristrut. dell'ER):

- ISA-SS[Sala] ∩ ISA-SC[Sala] = ∅
- Sala ⊆ ISA-BS[Sala] ∪ ISA-BC[Sala]
- Giud[Ric,Sala] ⊆ Per[Ric,Sala]

Schema logico: traduzione diretta (cont.1)

Regione(codice, numAbitanti)

Sala(codice, codRegione, annoInn)
fk: Sala[codRegione] ⊆ Regione[codice]

SenzaMensa(codice,codRegione, distRist)
fk: SenzaMensa[codice,codRegione] ⊆ Sala[codice,codRegione]

ConMensa(codice,codRegione,numPosti)
fk: ConMensa[codice,codRegione] ⊆ Sala[codice,codRegione]

Vincoli per catturare generalizzazione completa:
SenzaMensa[codice,codRegione] ∩ ConMensa[codice,codRegione] = ∅
Sala[codice,codRegione] ⊆ SenzaMensa[codice,codRegione] ∪ ConMensa [codice,codRegione]

Universita'(nome,dataIn)
fk: Universita'[nome] ⊆ Sede[nomeUniversita']

Sede(nomeUniversita', codRegione)
fk: Sede[nomeUniversita'] ⊆ Universita'[nome]
fk: Sede[codRegione] ⊆ Regione[codice]

…

Schema logico: traduzione diretta (cont.2)

…

Richiesta(nomeUniversita', codSala, codRegione,data)
fk: Richiesta[nomeUniversita'] ⊆ Universita'[nome]
fk: Richiesta[codSala, codRegione] ⊆ Sala[codice, codRegione]

Accolta(nomeUniversita', codSala, codRegione, dataRichiesta, dataAccoglimento)
fk: Accolta[nomeUniversita', codSala, codRegione, dataRichiesta] ⊆
Richiesta[nomeUniversita', codSala, codRegione,data]

Sfruttata(nomeUniversita', codSala, codRegione, dataRichiesta)
fk: Sfruttara[nomeUniversita',codSala,codRegione,dataRichiesta] ⊆
Accolta[nomeUniversita',codSala,codRegione,dataRichiesta]

…

Schema logico: traduzione diretta (cont.3)

…

Giud(nomeUniversita', codSala, codRegione, dataRichiesta, codSala2, codRegione2, giudizio)
fk: Giud[nomeUniversita', codSala, codRegione, dataRichiesta] ⊆
Sfruttata[nomeUniversita', codSala, codRegione, dataRichiesta]
fk: Giud[codSala2, codRegione2] ⊆ ConMensa[codice, codRegione]

Inoltre dobbiamo esprimere come vincolo aggiuntivo l'isa tra le relazioni "Giud" e "Per".
Per farlo dobbiamo ricostruire la relazione "Per" dalla relazione "Sfruttata" con una vista:

```
create view Per(nomeUniversita', codSala, codRegione, dataRichiesta, codSala2, codRegione2) as
  select nomeUniversita', codSala, codRegione, dataRichiesta,
         codSala as codSala2, codRegione as codRegione2
  from Sfruttata
```

A questo punto possiamo esprimere l'inclusione:

inc: Giud[nomeUniversita', codSala, codRegione, dataRichiesta, codSala2,codRegione2] ⊆
Per[nomeUniversita', codSala, codRegione, dataRichiesta, codSala2,codRegione2]

Tuttavia a questo punto notiamo che codSala2 e codRegione2 sono delle semplici repliche di codSala e codRegione, quindi possiamo riscrivere la relazione "Giud" in modo equivalente come segue:

Giud(nomeUniversita', codSala, codRegione, dataRichiesta, giudizio)
fk: Giud[nomeUniversita', codSala, codRegione, dataRichiesta] ⊆
Sfruttata[nomeUniversita', codSala, codRegione, dataRichiesta]
fk: Giud[codSala, codRegione] ⊆ ConMensa[codice, codRegione]

senza ulteriori vincoli.

Problema 2: Ristrutturazione schema logico

Si richiede di effettuare la **progettazione logica** dell'applicazione, producendo (in qualunque forma) lo schema relazionale completo di vincoli, seguendo l'indicazione che quando si accede ai dati relativi ad una università si vuole spesso conoscere la regione in cui ha sede.

Ristrutturazione schema logico

La parte dello schema logico interessata dalla ristrutturazione e' *(il resto dello schema logico rimane immutato)*:

Universita'(nome,dataIn)
fk: Universita'[nome] ⊆ Sede[nomeUniversita']

Sede(nomeUniversita', codRegione)
fk: Sede[nomeUniversita'] ⊆ Universita'[nome]
fk: Sede[codRegione] ⊆ Regione[codice]

Le due relazioni vanno **accorpate**, ottenendo:

Universita'(nome,dataIn,codRegione)
fk: Universita'[codRegione] ⊆ Regione[codice]

Problema 3

Sia dato il seguente schema relazionale

- Programmatore(Nome,Stipendio)
- Linguaggio(Nome,AnnoRilascio)
- Conosce(NomeProgrammatore,NomeLinguaggio,daAnno)

dove la tabella Programmatore rappresenta programmatori con nome e stipendio; la tabella Linguaggio rappresenta linguaggi con nome e anno di rilascio del linguaggio stesso; la tabella Conosce contiene tuple che rappresentano che un dato programmatore conosce un dato linguaggio da un dato anno. Si risponda alle seguenti query:

1. Restituire il nome e stipendio programmatori che conoscono almeno un linguaggio dall'anno del suo rilascio.
2. Restituire i nome dei programmatori che non conoscono alcun linguaggio il cui anno di rilascio e' successivo al 2000.
3. Restituire di ogni programmatore che conosce almeno 3 linguaggi, il numero di linguaggi da esso conosciuti il cui anno di rilascio è successivo al 2000.

Query 1

Restituire il nome e stipendio programmatori che conoscono almeno un linguaggio dall'anno del suo rilascio

```
select c.nome, c.stipendio
form Conosce c, Programmatore p, Linguaggio l
where c.nomeProgrammatore = p.nome and c.nomeLinguaggio = l.nome and
        c.daAnno=l.annoRilascio
```

Query 2

*Restituire i nome dei programmatori che **non** conoscono alcun linguaggio il cui anno di rilascio e' successivo al 2000.*

```
select nome
from Programmatore
  except
select c.nomeProgrammatore
from Conosce c,Linguaggio l
where c.nomeLinguaggio = l.nome and
        l.annoRilascio > 2000
```

Query 3

Restituire di ogni programmatore che conosce almeno 3 linguaggi, il numero di linguaggi da esso conosciuti il cui anno di rilascio è successivo al 2000.

```
create view ProgrammatoriCheConosconoAlmeno3Linguaggi as
select nomeProgrammatore
from Conosce
group by nomeProgrammatore
having count(*) >= 3

select c.nomeProgrammatore, count(*)
from Conosce c, Linguaggio l, ProgrammatoriCheConosconoAlmeno3Linguaggi p
where c.nomeProgrammatore = p.nomeProgrammatore and
        c.nomeLinguaggio = l.Nome
        and l.annoRilascio > 2000
group by c.nomeProgrammatore
```

Si noti che se la query fosse stata: *Restituire per ogni programmatore che conosce almeno 3 linguaggi, il numero di linguaggi da esso conosciuti; la query SQL non avrebbe richiesto di definire una vista:*

```
select c.nomeProgrammatore, count(*)
from Conosce c, Linguaggio l, ProgrammatoriCheConosconoAlmeno3Linguaggi p
where c.nomeProgrammatore = p.nomeProgrammatore and
        c.nomeLinguaggio = l.Nome
        and l.annoRilascio > 2000
group by c.nomeProgrammatore
having count(*) >= 3
```

Query aggiuntive

E' interessante vedere alcune varianti della query 2

*Restituire i nome dei programmatori che **non conoscono alcun linguaggio** il cui anno di rilascio e' successivo al 2000.*

Si noti che queste varianti non erano parte del compito del 04/12/06.

*Restituire i nome dei programmatori che **non conoscono almeno un linguaggio** il cui anno di rilascio e' successivo al 2000.*

```
                                        -- Vista che calcola il prodotto cartesiano:
create view NonConosce2000 as  -- tutte le coppie progr., ling. con anno di rilascio > 2000
select p.nome as nomeProgrammatore, l.nome as nomeLinguaggio
from Programmatore p, Linguaggio l
where l.annoRilascio > 2000
  except
select c.nomeProgrammatore, l.nomeLinguaggio
from Conosce c

select nomeProgrammatore
from NonConosce2000
```

*Restituire i nome dei programmatori che **conoscono tutti i linguaggi** il cui anno di rilascio e' successivo al 2000.*

```
select Nome
from Programmatore
  except
select NomeProgrammatore
from NonConosce2000
```