Basi di dati

Appello del 04-12-2006

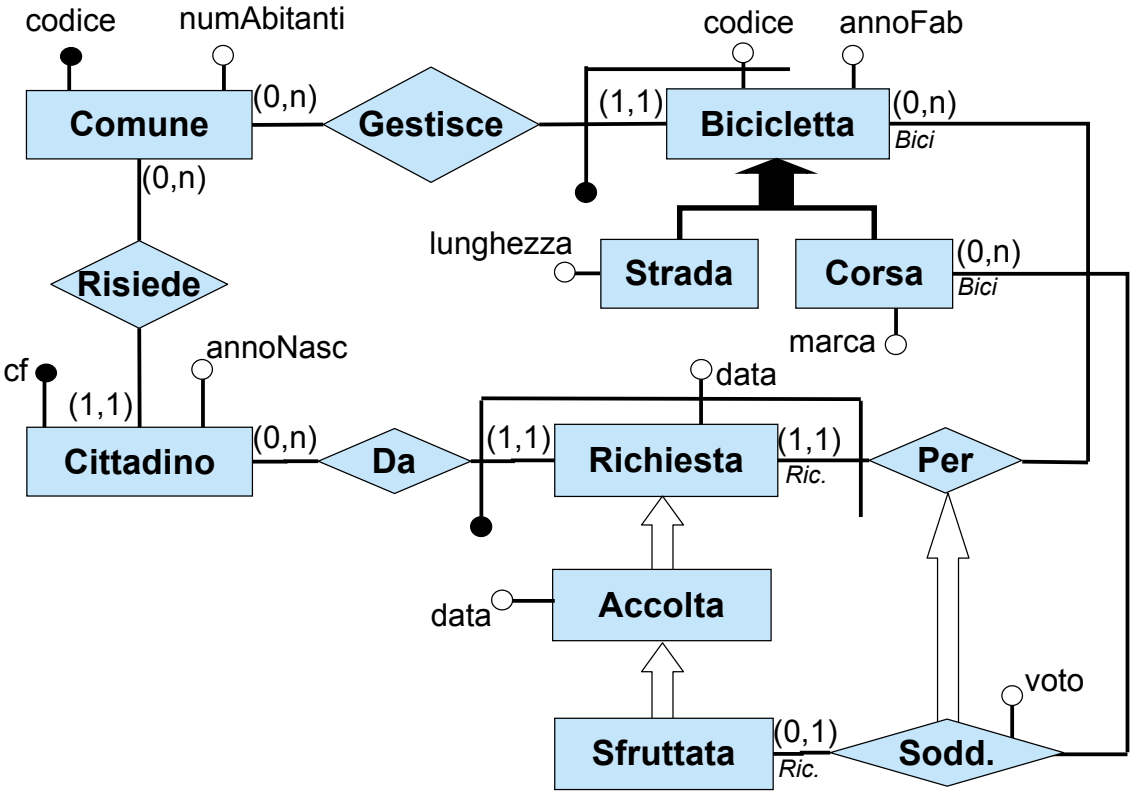
Soluzione Compito A

Anno Accademico 2006/07

Problema 1

Si richiede di **progettare lo schema concettuale Entità-Relazione** di un'applicazione relativa alla gestione delle biciclette date in uso dai comuni ai cittadini. Di ogni comune interessa il codice (identificativo) ed il numero di abitanti. Si noti che esistono comuni che sono di interesse per la nostra applicazione ma che attualmente non hanno alcuna bicicletta in gestione. Di ogni bicicletta interessa il comune responsabile della gestione, il codice (che identifica la bicicletta nell'ambito del comune responsabile) e l'anno di fabbricazione. Esistono esattamente due tipi di biciclette: da corsa e da strada. Delle prime interessa la marca, e delle seconde interessa la lunghezza. Di ogni cittadino interessa il codice fiscale (identificativo), la data di nascita, ed il comune di residenza. All'applicazione interessano le richieste di uso di bicicletta presentate dai cittadini. Ogni richiesta è presentata da esattamente un cittadino e riguarda l'uso di esattamente una bicicletta in una certa data. Alcune delle richieste sono accolte, e di esse interessa la data di accoglimento. Solo per le richieste che riguardano le biciclette da corsa e che sono state accolte e poi sfruttate dal cittadino (cioè a fronte delle quali il cittadino ha poi effettivamente utilizzato la bicicletta), il cittadino stesso ha la possibilità (non l'obbligo) di esprimere un voto di soddisfazione (numero da 0 a 10), e tale voto è di interesse per l'applicazione.

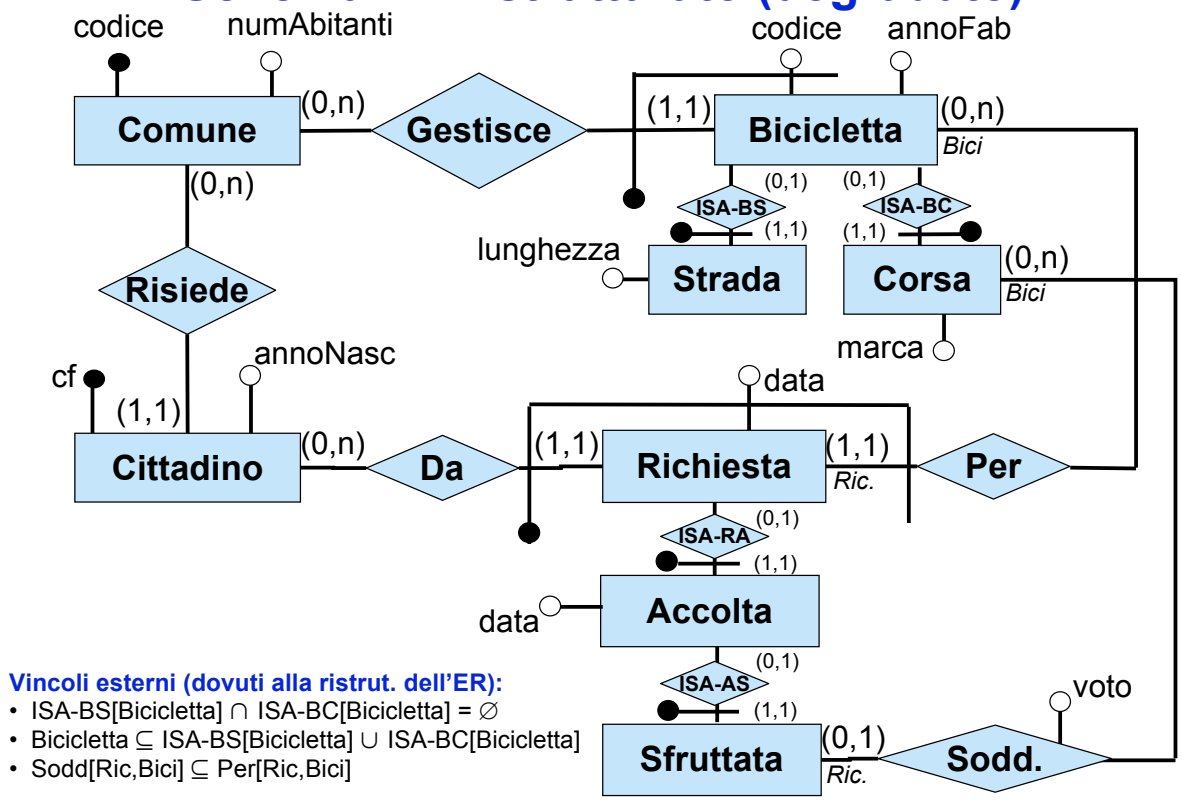
Schema ER



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 un cittadino si vuole spesso conoscere il comune in cui risiede.

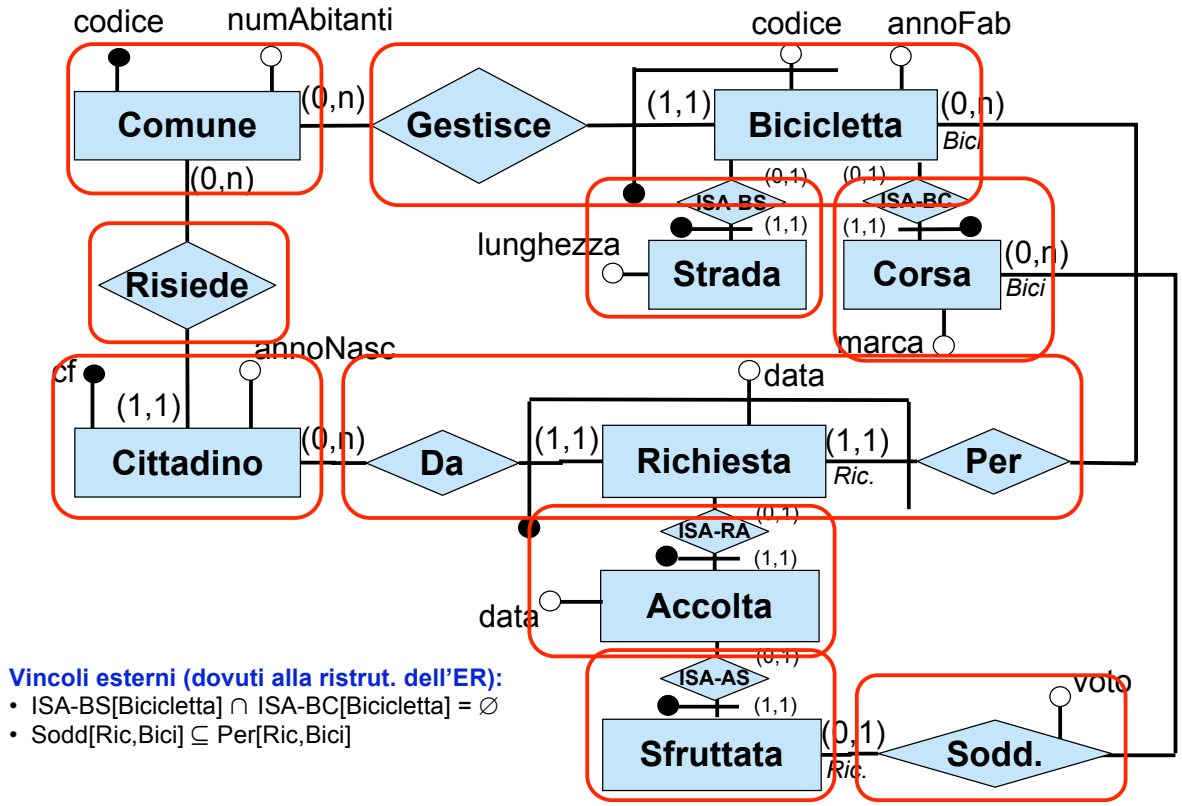
Schema ER ristrutturato (degradato)



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

- ISA-BS[Bicicletta] ∩ ISA-BC[Bicicletta] = ∅
- Bicicletta ⊆ ISA-BS[Bicicletta] ∪ ISA-BC[Bicicletta]
- Sodd[Ric,Bici] ⊆ Per[Ric,Bici]

Schema logico: traduzione diretta del ER



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

- ISA-BS[Bicicletta] ∩ ISA-BC[Bicicletta] = ∅
- Sodd[Ric,Bici] ⊆ Per[Ric,Bici]

Schema logico: traduzione diretta (cont.1)

Comune(codice, numAbitanti)

Bicicletta(codice, codComune, annoFab)
fk: Bicicletta[codComune] ⊆ Comune[codice]

Strada(codice,codComune, lunghezza)
fk: Strada[codice,codComune] ⊆ Bicicletta[codice,codComune]

Corsa(codice,codComune,marca)
fk: Corsa[codice,codComune] ⊆ Bicicletta[codice,codComune]

Vincoli per catturare generalizzazione completa:
Strada[codice,codComune] ∩ Corsa[codice,codComune] = ∅
Bicicletta[codice,codComune] ⊆ Strada[codice,codComune] ∪ Corsa [codice,codComune]

Cittadino(cf,annoNascita)
fk: Cittadino[cf] ⊆ Risiede[cfCittadino]

Risiede(cfCittadino, codComune)
fk: Risiede[cfCittadino] ⊆ Cittadino[cf]
fk: Risiede[codComune] ⊆ Comune[codice]

…

Schema logico: traduzione diretta (cont.2)

…

Richiesta(cfCittadino, codBicicletta, codComune,data)
fk: Richiesta[cfCittadino] ⊆ Cittadino[cf]
fk: Richiesta[codBicicletta, codComune] ⊆ Bicicletta[codice, codComune]

Accolta(cfCittadino, codBicicletta, codComune, dataRichiesta, dataAccoglimento)
fk: Accolta[cfCittadino, codBicicletta, codComune, dataRichiesta] ⊆
Richiesta[cfCittadino, codBicicletta, codComune,data]

Sfruttata(cfCittadino, codBicicletta, codComune, dataRichiesta)
fk: Sfruttara[cfCittadino,codBicicletta,codComune,dataRichiesta] ⊆
Accolta[cfCittadino,codBicicletta,codComune,dataRichiesta]

…

Schema logico: traduzione diretta (cont.3)

…
Soddisfazione(cfCittadino, codBicicletta, codComune, dataRichiesta, codBicicletta2, codComune2, voto)
fk: Soddisfazione[cfCittadino, codBicicletta, codComune, dataRichiesta] ⊆
Sfruttata[cfCittadino, codBicicletta, codComune, dataRichiesta]
fk: Soddisfazione[codBicicletta2, codComune2] ⊆ Corsa[codice, codComune]

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

```
create view Per(cfCittadino, codBicicletta, codComune, dataRichiesta, codBicicletta2, codComune2) as
  select cfCittadino, codBicicletta, codComune, dataRichiesta,
         codBicicletta as codBicicletta2, codComune as codComune2
  from Sfruttata
```

A questo punto possiamo esprimere l'inclusione:

inc: Soddisfazione[cfCittadino, codBicicletta, codComune, dataRichiesta, codBicicletta2,codComune2] ⊆
Per[cfCittadino, codBicicletta, codComune, dataRichiesta, codBicicletta2,codComune2]

Tuttavia a questo punto notiamo che codBicicletta2 e codComune2 sono delle semplici repliche di codBicicletta e codComune, quindi possiamo riscrivere la relazione "Soddisfazione" in modo equivalente come segue:

Soddisfazione(cfCittadino, codBicicletta, codComune, dataRichiesta, voto)
fk: Soddisfazione[cfCittadino, codBicicletta, codComune, dataRichiesta] ⊆
Sfruttata[cfCittadino, codBicicletta, codComune, dataRichiesta]
fk: Soddisfazione[codBicicletta, codComune] ⊆ Corsa[codice, codComune]

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 un cittadino si vuole spesso conoscere il comune in cui risiede.

Ristrutturazione schema logico

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

Cittadino(cf,annoNascita)
fk: Cittadino[cf] ⊆ Risiede[cfCittadino]

Risiede(cfCittadino, codComune)
fk: Risiede[cfCittadino] ⊆ Cittadino[cf]
fk: Risiede[codComune] ⊆ Comune[codice]

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

Cittadino(cf,annoNascita,codComune)
fk: Cittadino[codComune] ⊆ Comune[codice]

Problema 3

Sia dato il seguente schema relazionale

- Impiegato(Nome,AnnoAssunzione)
- Progetto(Nome,Durata)
- Lavora(NomeImpiegato,NomeProgetto,daAnno)

dove la tabella Impiegato rappresenta impiegati con nome e anno di assunzione; la tabella Progetto rappresenta progetti con nome e durata (in anni); la tabella Lavora contiene tuple che rappresentano che un dato impiegato lavora ad un dato progetto da un dato anno. Si risponda alle seguenti query:

1. Restituire il nome e la durata dei progetti sui quali lavora almeno un impiegato dall'anno della sua assunzione.
2. Restituire i nome dei progetti a cui non lavorano impiegati assunti dopo del 1980.
3. Restituire per ogni progetto al quale lavorano almeno 10 impiegati, il numero di impiegati che sono stati assunti dopo il 1980.

Query 1

Restituire il nome e la durata dei progetti sui quali lavora almeno un impiegato dall'anno della sua assunzione.

```
select p.nome, p.durata
form Lavora l, Progetto p, Impiegato i
where l.nomeProgetto = p.nome and l.nomeImpiegato = i.nome and
        i.daAnno=l.annoAssunzione
```

Query 2

*Restituire i nome dei progetti a cui **non** lavorano impiegati assunti dopo del 1980.*

```
select Nome
from Progetto
  except
select l.nomeProgetto
from Lavora l,Impiegati i
where l.nomeImpiegato = i.nome and
        i.annoAssunzione > 1980
```

Query 3

Restituire per ogni progetto al quale lavorano almeno 10 impiegati, il numero di impiegati che sono stati assunti dopo il 1980.

```
create view ProgettiConAlmento10Impiegati as
select nomeProgetto
from Lavora
group by nomeProgetto
having count(*) >= 10

select l.nomeProgetto, count(*)
from Lavora l, Impiegato i, ProgettiConAlmeno10Impiegati p
where l.nomeProgetto = p.nomeProgetto and
        l.nomeImpiegato = i.nome
        and i.annoAssumzione > 1980
group by l.nomeProgetto
```

Si noti che se la query fosse stata: *Restituire per ogni progetto al quale lavorano almeno 10 impiegati assunti dopo il 1980, il numero di impiegati; la query SQL non avrebbe richiesto di definire una vista:*

```
select l.nomeProgetto, count(*)
from Lavora l, Impiegato i, ProgettiConAlmeno10Impiegati p
where l.nomeProgetto = p.nomeProgetto and
        l.nomeImpiegato = i.nome
        and i.annoAssumzione > 1980
group by l.nomeProgetto
having count(*) >= 10
```

Query aggiuntive

E' interessante vedere alcune varianti della query 2

*Restituire i nome dei progetti a cui **non lavorano impiegati** assunti dopo del 1980.*

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

*Restituire i nome dei progetti a cui **non lavora almeno un impiegato** il cui anno di assunzione e' successivo al 1980.*

```
                                    -- Vista che calcola il prodotto cartesiano:
create view NonLavora1980 as        -- tutte le coppie progetto,impiegato con anno di assunzione > 1980
select p.nome as nomeProgetto, i.nome as nomeImpiegato
from Progetto p, Impiegato i
where i.annoAssunzione > 1980
  except
select l.nomeProgetto, l.nomeImpiegato
from Lavora l

select NomeProgrammatore
from NonLavora1980
```

*Restituire i nome dei progetti per cui lavorano **tutti gli impiegati** il cui anno di assunzione e' successivo al 1980.*

```
select nome
from Progetto
  except
select nomeProgetto
from NonLavora1980
```