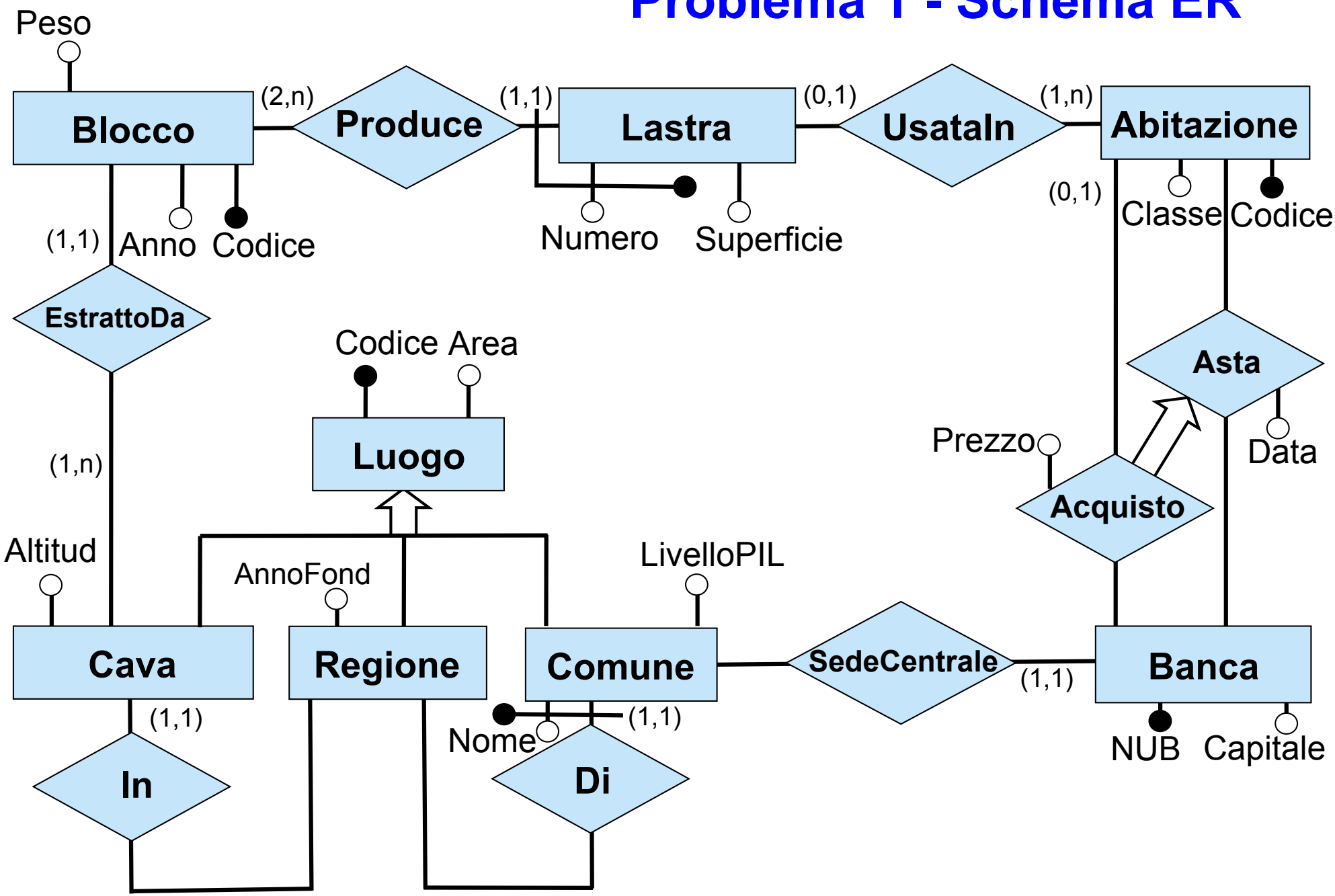
Basi di dati

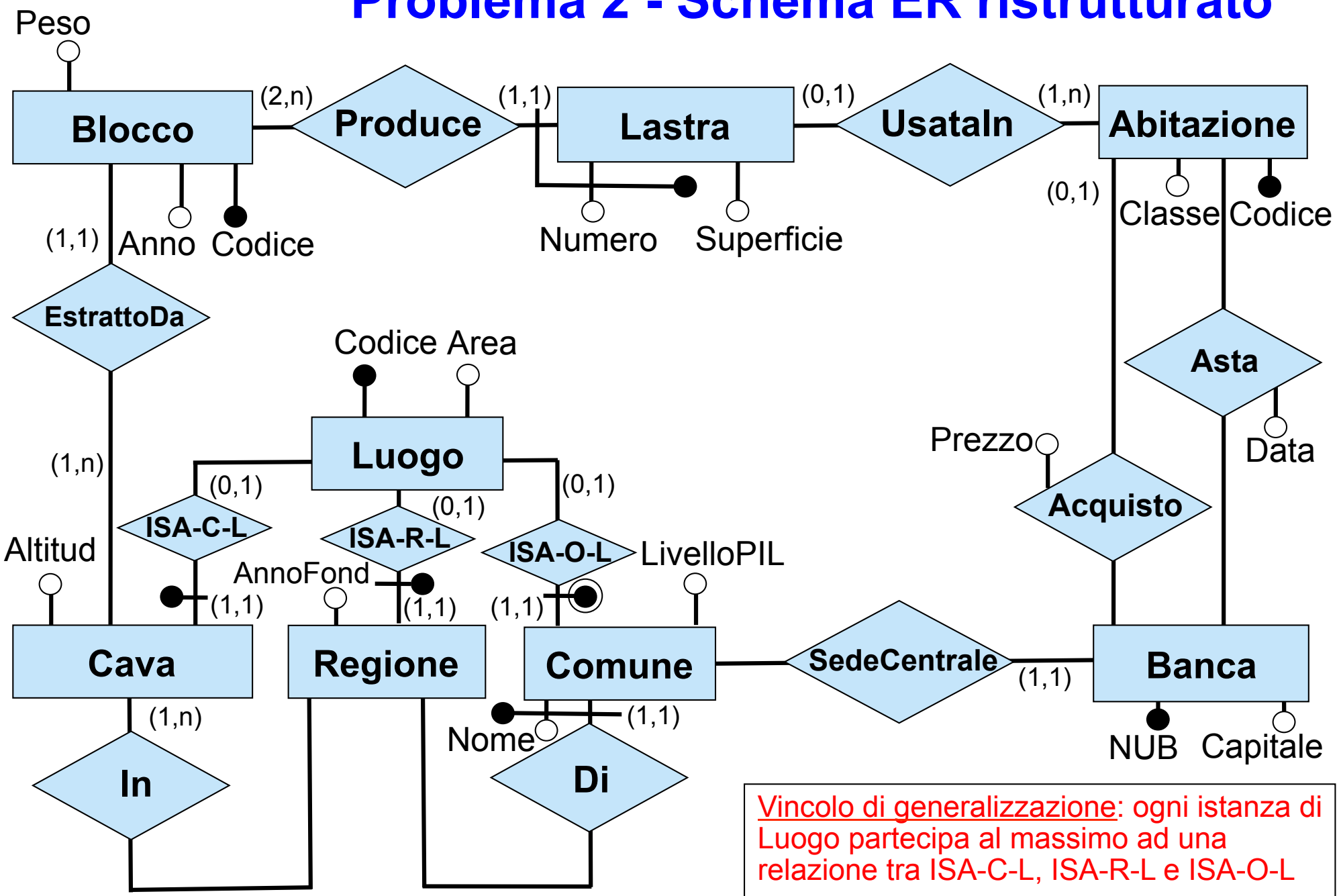
Appello del 4-02-2011
Compito B

Anno Accademico 2010/11

Problema 1 - Schema ER



Problema 2 - Schema ER ristrutturato



Vincolo di generalizzazione: ogni istanza di Luogo partecipa al massimo ad una relazione tra ISA-C-L, ISA-R-L e ISA-O-L

Vincolo esterno: ogni istanza di Acquisto è istanza di Asta

Problema 2 - Schema logico dalla traduzione

Blocco(codice, anno, peso)

foreign key: Blocco[codice] ⊆ EstrattoDa[blocco]

EstrattoDa(blocco, cava)

foreign key: EstrattoDa[blocco] ⊆ Blocco[codice]

foreign key: EstrattoDa[cava] ⊆ Cava[codice]

Abitazione(codice, classe)

foreign key: Abitazione[codice] ⊆ UsataIn[abitazione]

Lastra(numero, blocco, superficie)

foreign key: Lastra[blocco] ⊆ Blocco[codice]

UsataIn(numLastra, bloccoLastra, abitazione)

foreign key: UsataIn[numLastra,bloccoLastra] ⊆ Lastra[numero,blocco]

foreign key: UsataIn[abitazione] ⊆ Abitazione[codice]

Acquisto(abitazione, banca, prezzo)

foreign key: Acquisto[abitazione,banca] ⊆ Asta[abitazione, banca]

Asta(abitazione, banca, data)

foreign key: Asta[abitazione] ⊆ Abitazione[codice]

foreign key: Asta[banca] ⊆ Banca[NUB]

SedeCentrale(banca,comune)

foreign key: SedeCentrale[banca] ⊆ Banca[CUB]

foreign key: SedeCentrale[comune] ⊆ Comune[codice]

Problema 2 - Schema logico dalla traduzione

Banca(CUB, capitale)

foreign key: Banca[CUB] ⊆ SedeCentrale[banca]

Comune(codice, nome, livelloPIL)

foreign key: Comune[codice] ⊆ Luogo[codice]

Comune[Codice] ∩ Regione[Codice] = ∅

Comune[Codice] ∩ Cava[Codice] = ∅

Di(comune, regione)

foreign key: Di[comune] ⊆ Comune[codice]

foreign key: Di[regione] ⊆ Regione[codice]

Regione(codice, annoFond)

foreign key: Regione[codice] ⊆ Luogo[codice]

Regione[Codice] ∩ Cava[Codice] = ∅

In(cava, regione)

foreign key: In[cava] ⊆ Cava[codice]

foreign key: In[regione] ⊆ Regione[codice]

Cava(codice, altitud)

foreign key: Cava[codice] ⊆ In[cava]

foreign key: Cava[codice] ⊆ Luogo[codice]

inclusione: Cava[codice] ⊆ EstrattoDa[cava]

Luogo(codice, area)

Problema 2 - Schema logico dalla traduzione

Vincolo 1: nell'equi-join tra "Comune" e "Di" calcolato con la condizione "Di.comune=Comune.codice", non esistono due tuple diverse con la stessa coppia (nome,regione), ovvero tale coppia è una chiave della relazione risultante dall'equi-join

Vincolo 2: nella "create table" relativa alla relazione "Blocco", occorre inserire la clausola:

```
check (2 <= ( select count(*)
              from Lastra
              where blocco = codice))
```

Problema 2 – Ristrutturazione schema logico

Per rispondere alla indicazione che quando si accede ad una blocco si vuole conoscere la cava dalla quale è stato estratto il blocco, occorre accorpare la relazione “Blocco” e la relazione “EstrattoDa” (e quindi modificare l’inclusione che prima era definita tra “Blocco” e “EstrattoDa” e l’inclusione che prima era definita tra “Cava” e “EstrattoDa”):

Blocco(codice, anno, peso, cava)

foreign key: Blocco[cava] $\subseteq$ Cava[codice]

Cava(codice, altitud)

inclusione: Cava[codice] $\subseteq$ Blocco[cava]

Problema 2 - Schema logico finale

Blocco(codice, anno, peso, cava)

foreign key: Blocco[codice] ⊆ Cava[codice]

Abitazione(codice, classe)

foreign key: Abitazione[codice] ⊆ UsataIn[abitazione]

Lastra(numero, blocco, superficie)

foreign key: Lastra[blocco] ⊆ Blocco[codice]

UsataIn(numLastra, bloccoLastra, abitazione)

foreign key: UsataIn[numLastra,bloccoLastra] ⊆ Lastra[numero,blocco]

foreign key: UsataIn[abitazione] ⊆ Abitazione[codice]

Acquisto(abitazione, banca, prezzo)

foreign key: Acquisto[abitazione,banca] ⊆ Asta[abitazione, banca]

Asta(abitazione, banca, data)

foreign key: Asta[abitazione] ⊆ Abitazione[codice]

foreign key: Asta[banca] ⊆ Banca[NUB]

Banca(CUB, capitale)

foreign key: Banca[CUB] ⊆ SedeCentrale[banca]

SedeCentrale(banca,comune)

foreign key: SedeCentrale[banca] ⊆ Banca[CUB]

foreign key: SedeCentrale[comune] ⊆ Comune[codice]

Problema 2 - Schema logico finale

Comune(<u>codice</u>, nome, livelloPIL)

foreign key: Comune[codice] ⊆ Luogo[codice]

Comune[Codice] ∩ Regione[Codice] = ∅

Comune[Codice] ∩ Cava[Codice] = ∅

Di(<u>comune</u>, regione)

foreign key: Di[comune] ⊆ Comune[codice]

foreign key: Di[regione] ⊆ Regione[codice]

Regione(<u>codice</u>, annoFond)

foreign key: Regione[codice] ⊆ Luogo[codice]

Regione[Codice] ∩ Cava[Codice] = ∅

In(<u>cava</u>, provincia)

foreign key: In[cava] ⊆ Cava[codice]

foreign key: In[regione] ⊆ Regione[codice]

Cava(<u>codice</u>, altitud)

foreign key: Cava[codice] ⊆ In[cava]

foreign key: Cava[codice] ⊆ Luogo[codice]

inclusione: Cava[codice] ⊆ Blocco[cava]

Luogo(<u>codice</u>, area)

Problema 2 - Schema logico finale

I vincoli rimangono immutati:

Vincolo 1: nell'equi-join tra "Comune" e "Di" calcolato con la condizione "Di.comune=Comune.codice", non esistono due tuple diverse con la stessa coppia (nome,regione), ovvero tale coppia è una chiave della relazione risultante dall'equi-join

Vincolo 2: nella "create table" relativa alla relazione "Blocco", occorre inserire la clausola:

```
check (2 <= ( select count(*)
              from Lastra
              where blocco = codice))
```

Problema 3

Si consideri uno schema relazionale in cui la relazione

Comune(Codice, Servizio, Anno, Spesa)

memorizza, per i vari comuni, quanto è stato speso nei vari anni per i vari servizi sociali, e la relazione

In(Codice, Regione)

specifica in quali regioni si trovano i comuni.

1. Per ogni comune della Toscana e per ogni anno, calcolare in quali servizi sociali il comune ha speso più di 100.000 euro.

***Soluzione**: È sufficiente un banale join tra "Comune" e "In".*

```
select Comune.Codice, Comune.Anno, Comune.Servizio
from  Comune, In
where Comune.Codice = In.Codice and
      In.Regione = 'Toscana' and Comune.Spesa > 100.000
```

Problema 3

2. Calcolare i servizi sociali per i quali tutte le spese effettuate nel 2010 sono stati effettuate da comuni di una sola regione.

***Soluzione**: Si calcola il complemento dell'insieme dell'insieme dei servizi sociali che nel 2010 hanno ottenuto incassi maggiori di 0 in comuni che si trovano in almeno due regioni diverse.*

```
select Servizio
from  Comune C
where C.Servizio not in
      ( select C1.Servizio
        from Comune C1, Comune C2, In R1, In R2
        where C1.Servizio = C2.Servizio and C1.anno = 2010
              and C2.Anno = 2010 and C1.Spesa > 0 and
              C2.Spesa > 0 and C1.Codice = R1.Codice and
              C2.Codice = R2.Codice and
              R1.Regione <> R2.Regione)
```

Problema 3

3. Per ogni anno e per ogni regione calcolare i servizi per i quali la somma delle spese effettuate dai comuni di quella regione in quell'anno supera 500.000 euro.

***Soluzione**: È sufficiente aggregare il join tra "Comune" e "In" su "Anno", "Regione" e "Servizio", ed usare la clausola "having" per filtrare solo i gruppi che hanno la spesa maggiore di 500.000.*

```
select C.Anno, R.Regione, C.Servizio
from  Comune C, In R
where C.Codice = R.Codice
group by C.Anno, R.Regione, C.Servizio
having sum(C.Spesa) > 500.000
```

Problema 4

Un vincolo di integrità è una condizione che si esprime a livello di schema della base di dati e che si intende debba essere soddisfatta da tutte le istanze della base di dati.

Il vincolo di integrità

> *Per tutti i comuni, il servizio sociale "assistenza malati AIDS" non compare tra quelli che hanno richiesto spese prima del 1975*

È un vincolo intra-relazionale, e più in particolare un vincolo di tupla, che si esprime in SQL mediante la seguente clausola "check" dentro la "create table" relativa alla tabella "Comune":

Create table Comune (…

…

```
Check (Servizio <> 'aids' or Anno >= 1975)
```

…

)