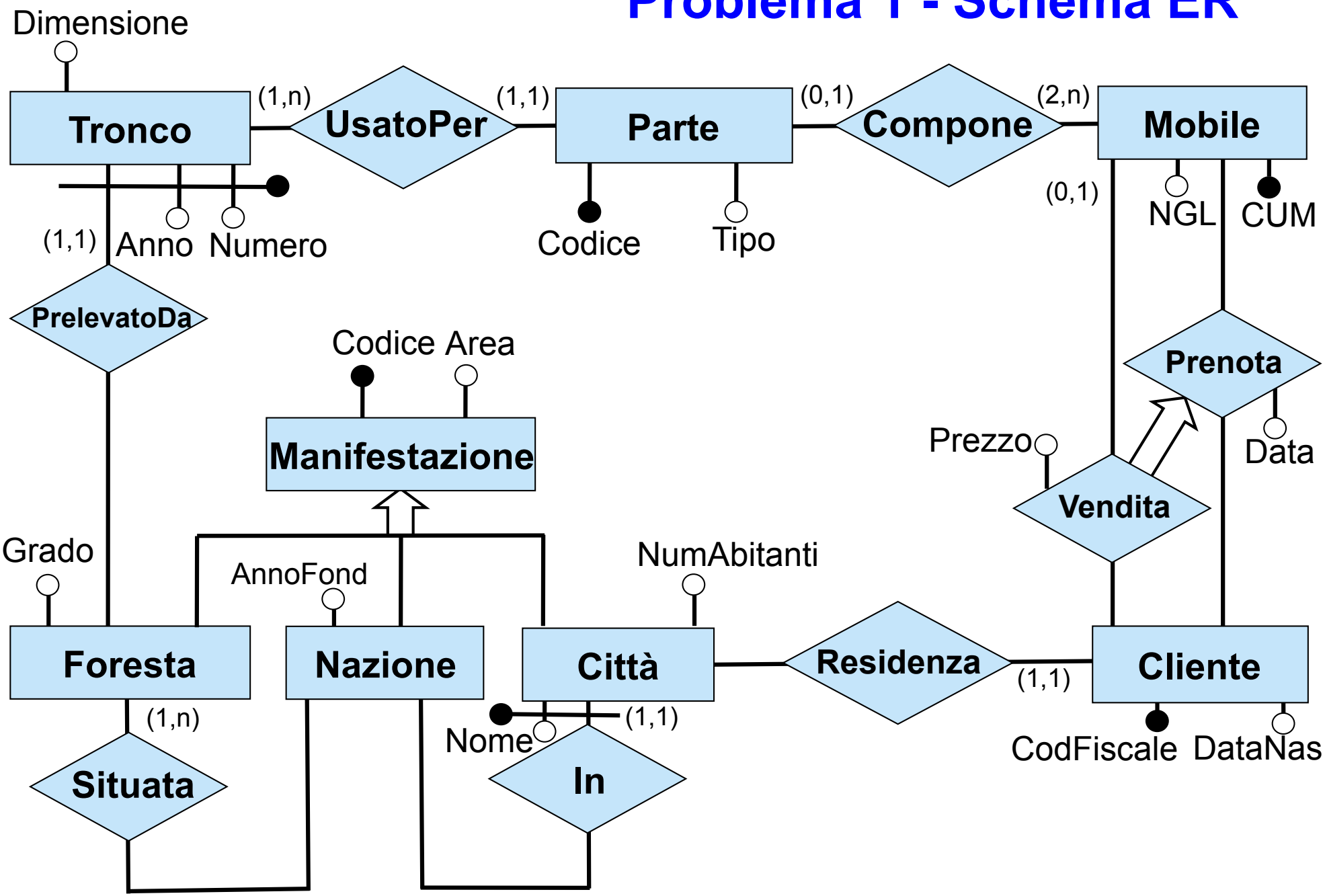
Basi di dati

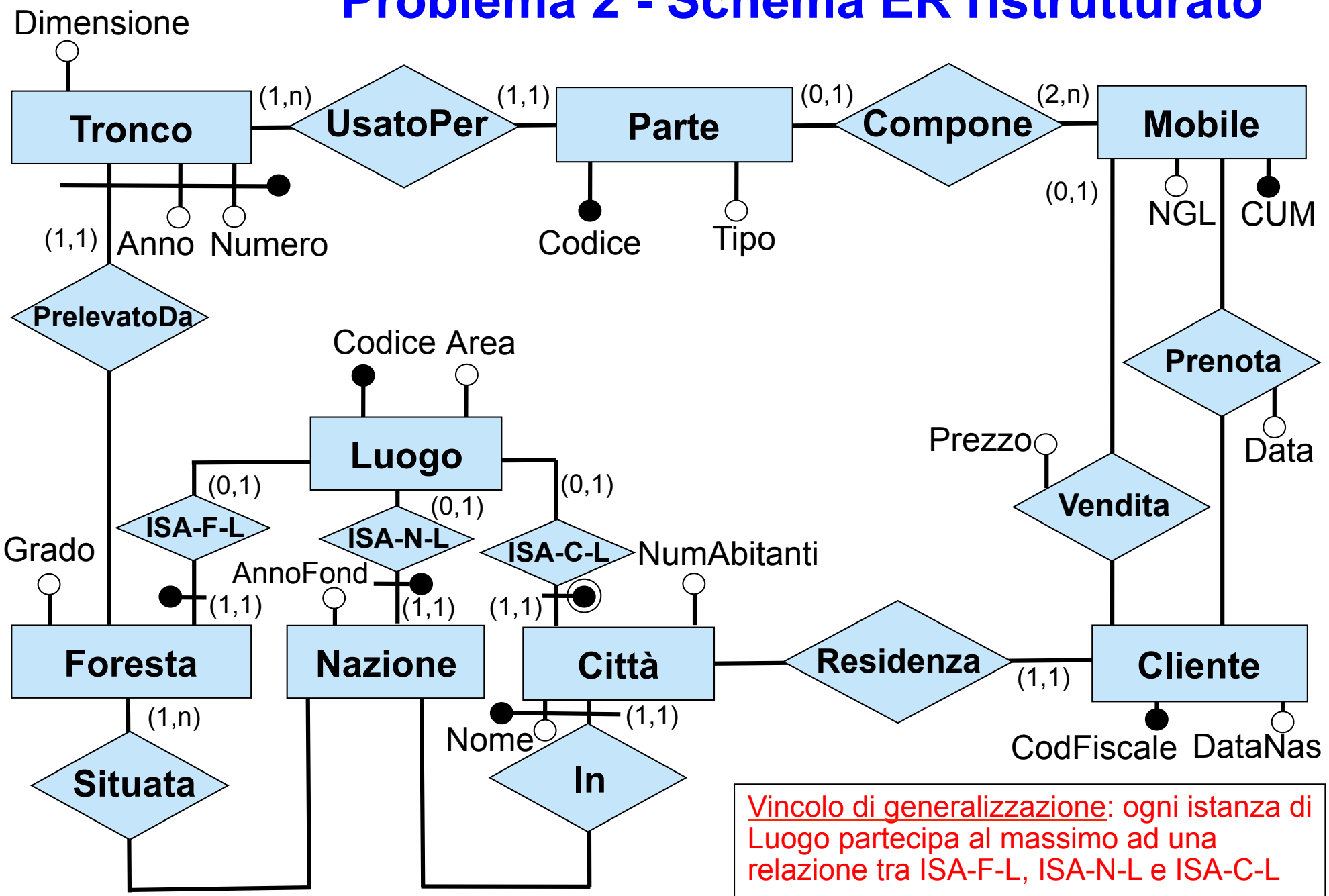
Appello del 23-02-2011

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-F-L, ISA-N-L e ISA-C-L

Vincolo esterno: ogni istanza di Vendita è istanza di Prenota

Problema 2 - Schema logico dalla traduzione

Tronco(numero, anno, foresta, dimensione)

inclusione: Tronco[numero,anno,foresta] ⊆ UsatoPer[numero,anno,foresta]

foreign key: Tronco[foresta] ⊆ Foresta[codice]

Parte(codice, tipo)

foreign key: Parte[codice] ⊆ UsatoPer[parte]

UsatoPer(parte, numero, anno, foresta)

foreign key: UsatoPer[parte] ⊆ Parte[codice]

foreign key: UsatoPer[numero,anno,foresta] ⊆ Tronco[numero,anno,foresta]

Compone(parte, mobile)

foreign key: Compone[parte] ⊆ Parte[codice]

foreign key: Compone[mobile] ⊆ Mobile[CUM]

Vendita(mobile, cliente, prezzo)

foreign key: Vendita[mobile,cliente] ⊆ Prenota[mobile,cliente]

Prenota(mobile, cliente, data)

foreign key: Prenota[mobile] ⊆ Mobile[CUM]

foreign key: Prenota[cliente] ⊆ Cliente[codFiscale]

Cliente(codFiscale, dataNas)

foreign key: Cliente[codFiscale] ⊆ Residenza[cliente]

Mobile(CUM,NGL)

Luogo(codice, area)

Problema 2 - Schema logico dalla traduzione

Residenza(cliente,città)

foreign key: Residenza[cliente] ⊆ Cliente[codFiscale]

foreign key: Residenza[città] ⊆ Città[codice]

Città(codice, nome, numAbitanti)

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

foreign key: Città[codice] ⊆ In[città]

Città[Codice] ∩ Nazione[Codice] = ∅

Città[Codice] ∩ Foresta[Codice] = ∅

In(città, nazione)

foreign key: In[città] ⊆ Città[codice]

foreign key: In[nazione] ⊆ Nazione[codice]

Nazione(codice, annoFond)

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

Nazione[Codice] ∩ Foresta[Codice] = ∅

Situata(foresta, nazione)

foreign key: Situata[foresta] ⊆ Foresta[codice]

foreign key: Situata[nazione] ⊆ Nazione[codice]

Foresta(codice, grado)

foreign key: Foresta[codice] ⊆ Situata[foresta]

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

Problema 2 - Schema logico dalla traduzione

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

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

```
check (2 <= ( select count(*)
              from Compone
              where mobile = CUM ))
```

Problema 2 – Ristrutturazione schema logico

Per rispondere alla indicazione che quando si accede ad una parte si vuole conoscere il tronco usato per produrla, occorre accorpare la relazione “Parte” e la relazione “UsatoPer” (e quindi modificare l’inclusione che prima era definita tra “Tronco” e “UsatoPer”):

Parte(<u>codice</u>, tipo, numero, anno, foresta)

foreign key: Parte[numero,anno,foresta] $\subseteq$ Tronco[numero,anno,foresta]

Tronco(<u>numero</u>, <u>anno</u>, <u>foresta</u>, dimensione)

inclusione: Tronco[numero,anno,foresta] $\subseteq$ Parte[numero,anno,foresta]

Problema 2 - Schema logico finale

Tronco(numero, anno, foresta, dimensione)

inclusione: Tronco[numero,anno,foresta] ⊆ Parte[numero,anno,foresta]

foreign key: Tronco[foresta] ⊆ Foresta[codice]

Parte(codice, tipo, numero, anno, foresta)

foreign key: Parte[numero,anno,foresta] ⊆ Tronco[numero,anno,foresta]

Compone(parte, mobile)

foreign key: Compone[parte] ⊆ Parte[codice]

foreign key: Compone[mobile] ⊆ Mobile[CUM]

Vendita(mobile, cliente, prezzo)

foreign key: Vendita[mobile,cliente] ⊆ Prenota[mobile,cliente]

Prenota(mobile, cliente, data)

foreign key: Prenota[mobile] ⊆ Mobile[CUM]

foreign key: Prenota[cliente] ⊆ Cliente[codFiscale]

Cliente(codFiscale, dataNas)

foreign key: Cliente[codFiscale] ⊆ Residenza[cliente]

Mobile(CUM,NGL)

Luogo(codice, area)

Problema 2 - Schema logico finale

Residenza(cliente,città)

foreign key: Residenza[cliente] ⊆ Cliente[codFiscale]

foreign key: Residenza[città] ⊆ Città[codice]

Città(codice, nome, numAbitanti)

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

foreign key: Città[codice] ⊆ In[città]

Città[Codice] ∩ Nazione[Codice] = ∅

Città[Codice] ∩ Foresta[Codice] = ∅

In(città, nazione)

foreign key: In[città] ⊆ Città[codice]

foreign key: In[nazione] ⊆ Nazione[codice]

Nazione(codice, annoFond)

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

Nazione[Codice] ∩ Foresta[Codice] = ∅

Situata(foresta, nazione)

foreign key: Situata[foresta] ⊆ Foresta[codice]

foreign key: Situata[nazione] ⊆ Nazione[codice]

Foresta(codice, grado)

foreign key: Foresta[codice] ⊆ Situata[foresta]

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

Problema 2 - Schema logico finale

I vincoli rimangono immutati:

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

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

```
check (2 <= ( select count(*)
              from Compone
              where mobile = CUM ))
```

Problema 3

Si consideri uno schema relazionale in cui la relazione

Palestra(Codice, Sport, Anno, Incasso)

memorizza, per le varie palestre, i vari anni ed i vari sport, gli incassi ottenuti, e la relazione

Dove(Codice, Città)

specifica in quali città si trovano le palestre.

1. Per ogni palestra di Pisa e per ogni anno, calcolare in quali sport la palestra ha ottenuto in quell'anno un incasso maggiore di 10.000 euro.

***Soluzione**: È sufficiente un banale join tra "Palestra" e "Dove".*

```
select Palestra.Codice, Palestra.Anno, Palestra.Sport
from  Palestra, Dove
where Palestra.Codice = Dove.Codice and
      Dove.Città = 'Pisa' and Palestra.Incasso > 10.000
```

Problema 3

2. Calcolare gli sport per i quali tutti gli incassi maggiori di 0 relativi al 2010 sono stati ottenuti da palestre di una sola città.

***Soluzione**: Si calcola il complemento dell'insieme degli sport che nel 2010 hanno ottenuto incassi maggiori di 0 in palestre che si trovano in almeno due città diverse.*

```
select Sport
from  Palestra P
where P.Sport not in
      ( select P1.Sport
        from Palestra P1, Palestra P2, Dove D1, Dove D2
        where P1.Sport = P2.Sport and P1.anno = 2010 and
              P2.Anno = 2010 and P1.Incasso > 0 and
              P2.Incasso > 0 and P1.Codice = D1.Codice and
              P2.Codice = D2.Codice and
              D1.Città <> D2.Città )
```

Problema 3

3. Per ogni anno e per ogni città calcolare gli sport per i quali la somma degli incassi ottenuti dalle palestre di quella città in quell'anno supera 100.000 euro.

***Soluzione**: È sufficiente aggregare il join tra "Palestra" e "Dove" su "Anno", "Città" e "Sport", ed usare la clausola "having" per filtrare solo i gruppi che hanno la somma degli incassi maggiore di 100.000.*

```
select P.Anno, D.Città, P.Sport
from  Palestra P, Dove D
where P.Codice = D.Codice
group by P.Anno, D.Città, P.Sport
having sum(P.Incasso) > 100.000
```

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 tutte le palestre, lo sport "climbing" non compare tra quelli che hanno prodotto incassi prima del 1970*

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 "Palestra":

```
create table Palestra (...
...
check (Sport <> 'climbing' or Anno >= 1970)
...
)
```