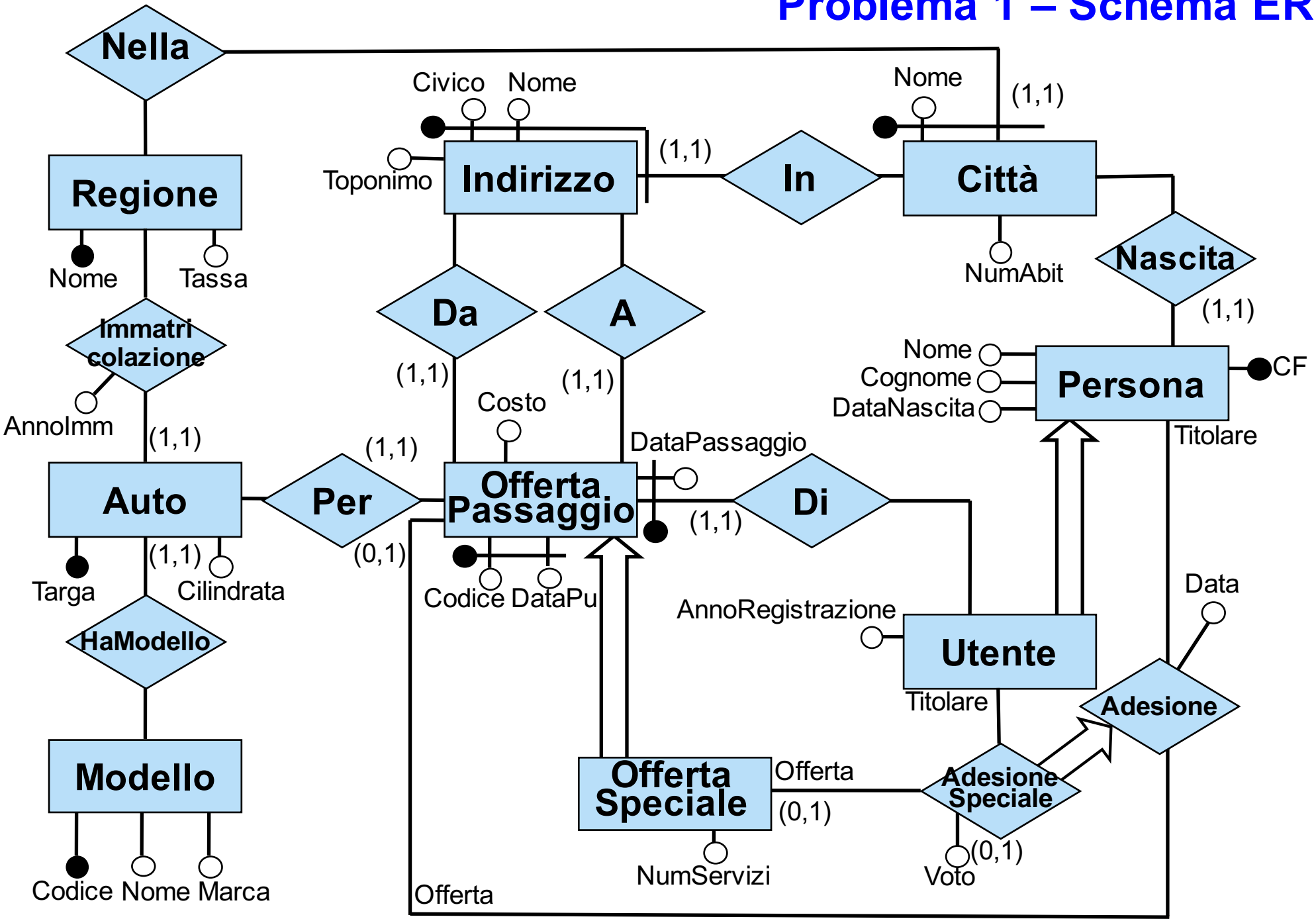
Basi di dati

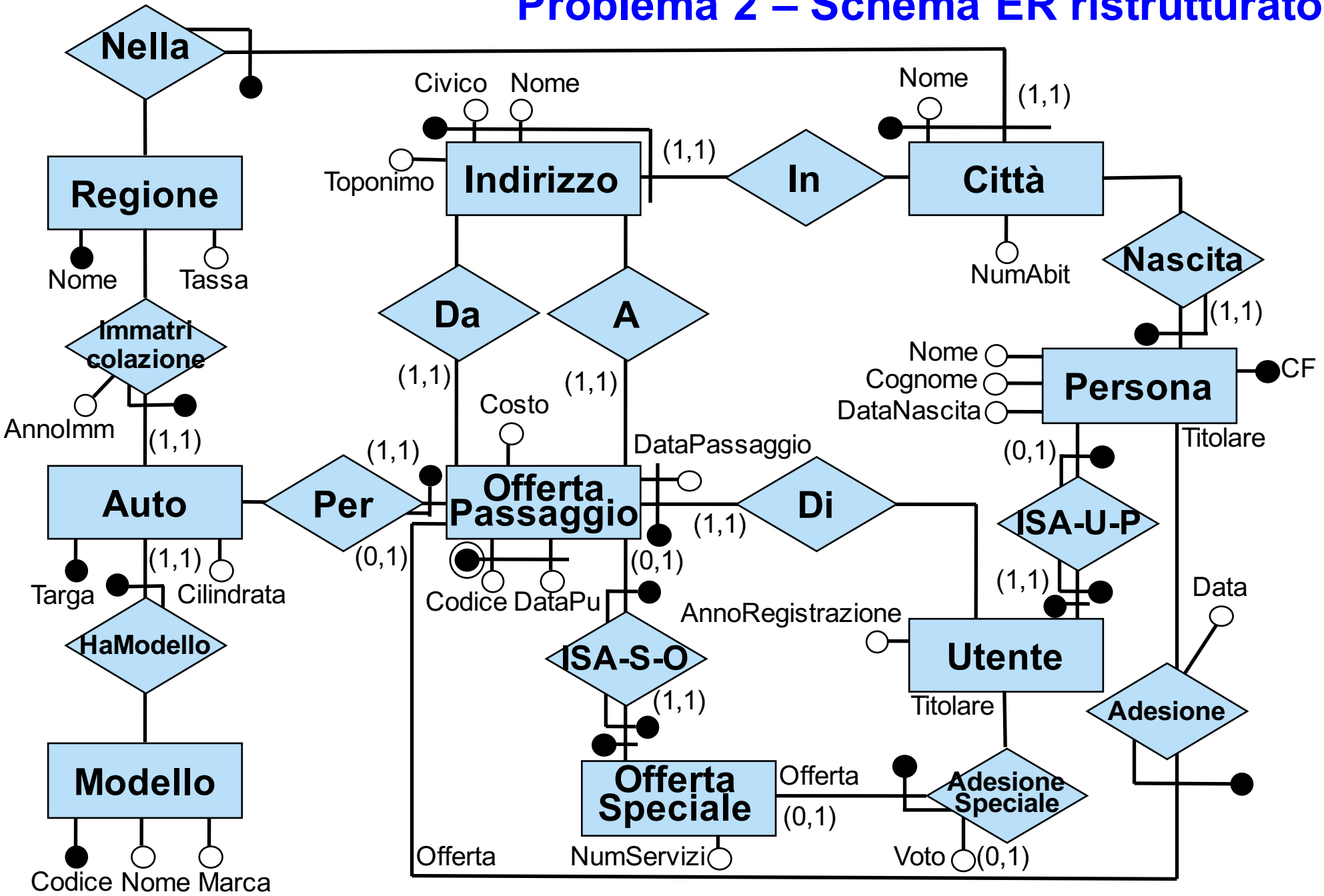
Appello del 8-01-2015
Compito A

Anno Accademico 2014/15

Problema 1 – Schema ER



Problema 2 – Schema ER ristrutturato



Vincolo esterno: Per ogni istanza <Offerta:o,Titolare:t> di AdesioneSpeciale, se o1 e t1 sono tali che <OffertaSpeciale:o,OffertaPassaggio:o1> è istanza di ISA-S-O e <Utente:t,Persona:t1> è istanza di ISA-U-P, si ha che l'stanza <Offerta:o1,Titolare:t1> è istanza di Adesione.

Problema 2 - Schema logico dopo la traduzione diretta

OffertaPassaggio(Codice, DataPu, DataPassaggio, Costo)
- foreign key: OffertaPassaggio[Codice,DataPu] ⊆ Per[CodiceOfferta,DataOfferta]
- foreign key: OffertaPassaggio[Codice,DataPu] ⊆ Da[CodiceOfferta,DataOfferta]
- foreign key: OffertaPassaggio[Codice,DataPu] ⊆ A[CodiceOfferta,DataOfferta]
- foreign key: OffertaPassaggio[Codice,DataPu] ⊆ Di[CodiceOfferta,DataOfferta]

Auto(Targa, Cilindrata)
- foreign key: Auto[Targa] ⊆ Immatricolazione[Auto]
- foreign key: Auto[Targa] ⊆ HaModello[Auto]

Per(CodiceOfferta, DataOfferta, Auto)
- foreign key: Per[CodiceOfferta,DataOfferta] ⊆ OffertaPassaggio[Codice,DataPu]
- foreign key: Per[Auto] ⊆ Auto[Targa]

Da(CodiceOfferta, DataOfferta, NomeInd, CivicoInd, NomeCittà, RegioneCittà)
- foreign key: Da[CodiceOfferta,DataOfferta] ⊆ OffertaPassaggio[Codice,DataPu]
- foreign key: Da[NomeInd,CivicoInd,NomeCittà,RegioneCittà] ⊆ Indirizzo[Nome,Civico,NomeCittà,RegioneCittà]

A(CodiceOfferta, DataOfferta, NomeInd, CivicoInd, NomeCittà, RegioneCittà)
- foreign key: A[CodiceOfferta,DataOfferta] ⊆ OffertaPassaggio[Codice,DataPu]
- foreign key: A[NomeInd,CivicoInd,NomeCittà,RegioneCittà] ⊆ Indirizzo[Nome,Civico,NomeCittà,RegioneCittà]

Indirizzo(Nome, Civico, NomeCittà, RegioneCittà, Toponimo)
- foreign key: Indirizzo[NomeCittà,NomeRegione] ⊆ Città[Nome,Regione]

Città(Nome, Regione, NumAbit)
- foreign key: Città[Regione] ⊆ Regione[Nome]

Regione(Nome, Tassa)

Modello(Codice, Nome, Marca)

HaModello(Auto,Modello)
- foreign key: HaModello[Auto] ⊆ Auto[Targa]
- foreign key: HaModello[Modello] ⊆ Modello[Codice]

Problema 2 - Schema logico dopo la traduzione diretta

Immatricolazione(Auto, Regione, AnnoImm)
foreign key: Immatricolazione[Auto] ⊆ Auto[Targa]
foreign key: Immatricolazione[Regione] ⊆ Regione[Nome]
Persona(CF, Nome, Cognome, DataNascita)
foreign key: Persona[CF] ⊆ Nascita[Persona]
Nascita(Persona, NomeCittà, RegioneCittà)
foreign key: Nascita[Persona] ⊆ Persona[CF]
foreign key: Nascita[NomeCittà,RegioneCittà] ⊆ Città[Nome,Regione]
Utente(CF, AnnoRegistrazione)
foreign key: Utente[CF] ⊆ Persona[CF]
Di(CodiceOfferta, DataOfferta, Utente)
foreign key: Di[CodiceOfferta,DataOfferta] ⊆ OffertaPassaggio[Codice,DataPu]
foreign key: Di[Utente] ⊆ Utente[CF]
Adesione(CodiceOfferta, DataOfferta, Titolare, Data)
foreign key: Adesione[CodiceOfferta,DataOfferta] ⊆ OffertaPassaggio[Codice,DataPu]
foreign key: Adesione[Titolare] ⊆ Persona[CF]
OffertaSpeciale(Codice, DataPu, NumServizi)
foreign key: OffertaSpeciale[Codice,DataPu] ⊆ OffertaPassaggio[Codice,DataPu]
AdesioneSpeciale(CodiceOfferta, DataOfferta, Titolare, Voto*)
foreign key: AdesioneSpeciale[CodiceOfferta,DataOfferta] ⊆ OffertaSpeciale[Codice,DataPu]
foreign key: AdesioneSpeciale[Titolare] ⊆ Utente[CF]
foreign key: AdesioneSpeciale[CodiceOfferta,DataOfferta,Titolare] ⊆
Adesione[CodiceOfferta,DataOfferta,Titolare]

Vincolo esterno: Nell'equi-join tra Di e OffertaPassaggio con condizione di join (CodiceOfferta=Codice AND DataOfferta=DataPu]), gli attributi DataPassaggio e Utente formano una chiave.

Problema 2 - Schema logico dopo la ristrutturazione

- *Indicazione 2*: Accorpamento di Auto e Immatricolazione (sfruttando l'accoppiamento forte)
- *Indicazione 3*: Accorpamento di OffertaPassaggio e Adesione (sfruttando l'accorpamento debole), considerando il valore nullo/non nullo come un flag, in modo che le tuple di OffertaPassaggio con Titolare <> NULL siano quelle il cui servizio è stato assegnato; si aggiunge anche il vincolo di tupla che impone che Titolare è nullo se e solo se DataAdesione è nullo.

OffertaPassaggio(Codice, DataPu, DataPassaggio, Costo, Titolare*, DataAdesione*)
  foreign key: OffertaPassaggio[Codice,DataPu] ⊆ Per[CodiceOfferta,DataOfferta]
  foreign key: OffertaPassaggio[Codice,DataPu] ⊆ Da[CodiceOfferta,DataOfferta]
  foreign key: OffertaPassaggio[Codice,DataPu] ⊆ A[CodiceOfferta,DataOfferta]
  foreign key: OffertaPassaggio[Codice,DataPu] ⊆ Di[CodiceOfferta,DataOfferta]
  foreign key: OffertaPassaggio[Titolare] ⊆ Persona[CF]
  vincolo di tupla: (Titolare <> NULL OR DataAdesione = NULL) AND
    (DataAdesione <> NULL OR Titolare = NULL)

Auto(Targa, Cilindrata, RegioneImm, AnnoImm)
  foreign key: Auto[Targa] ⊆ HaModello[Auto]

Per(CodiceOfferta, DataOfferta, Auto)
  foreign key: Per[CodiceOfferta,DataOfferta] ⊆ OffertaPassaggio[Codice,DataPu]
  foreign key: Per[Auto] ⊆ Auto[Targa]

Da(CodiceOfferta, DataOfferta, NomeInd, CivicoInd, NomeCittà, RegioneCittà)
  foreign key: Da[CodiceOfferta,DataOfferta] ⊆ OffertaPassaggio[Codice,DataPu]
  foreign key: Da[NomeInd,CivicoInd,NomeCittà,RegioneCittà] ⊆
    Indirizzo[Nome,Civico,NomeCittà,RegioneCittà]

A(CodiceOfferta, DataOfferta, NomeInd, CivicoInd, NomeCittà, RegioneCittà)
  foreign key: A[CodiceOfferta,DataOfferta] ⊆ OffertaPassaggio[Codice,DataPu]
  foreign key: A[NomeInd,CivicoInd,NomeCittà,RegioneCittà] ⊆
    Indirizzo[Nome,Civico,NomeCittà,RegioneCittà]

Problema 2 - Schema logico dopo la ristrutturazione

Indirizzo(Nome, Civico, NomeCittà, RegioneCittà, Toponimo)
foreign key: Indirizzo[NomeCittà,NomeRegione] ⊆ Città[Nome,Regione]
Città(Nome, Regione, NumAbit)
foreign key: Città[Regione] ⊆ Regione[Nome]
Regione(Nome, Tassa)
Modello(Codice, Nome, Marca)
HaModello(Auto,Modello)
foreign key: HaModello[Auto] ⊆ Auto[Targa]
foreign key: HaModello[Modello] ⊆ Modello[Codice]
Persona(CF, Nome, Cognome, DataNascita)
foreign key: Persona[CF] ⊆ Nascita[Persona]
Nascita(Persona, NomeCittà, RegioneCittà)
foreign key: Nascita[Persona] ⊆ Persona[CF]
foreign key: Nascita[NomeCittà,RegioneCittà] ⊆ Città[Nome,Regione]
Utente(CF, AnnoRegistrazione)
foreign key: Utente[CF] ⊆ Persona[CF]
Di(CodiceOfferta, DataOfferta, Utente)
foreign key: Di[CodiceOfferta,DataOfferta] ⊆ OffertaPassaggio[Codice,DataPu]
foreign key: Di[Utente] ⊆ Utente[CF]
OffertaSpeciale(Codice, DataPu, NumServizi)
foreign key: OffertaSpeciale[Codice,DataPu] ⊆ OffertaPassaggio[Codice,DataPu]
AdesioneSpeciale(CodiceOfferta, DataOfferta, Titolare, Voto*)
foreign key: AdesioneSpeciale[CodiceOfferta,DataOfferta] ⊆ OffertaSpeciale[Codice,DataPu]
foreign key: AdesioneSpeciale[Titolare] ⊆ Utente[CF]
foreign key: AdesioneSpeciale[CodiceOfferta,DataOfferta,Titolare] ⊆
Adesione[CodiceOfferta,DataOfferta,Titolare]

Vincolo esterno: Nell'equi-join tra Di e OffertaPassaggio con condizione di join (CodiceOfferta=Codice AND DataOfferta=DataPu]), gli attributi DataPassaggio e Utente formano una chiave.

Problema 3 - Query 1

Testo: Per ogni cane san bernardo femmina, calcolare il nome, l'anno di nascita e la città di nascita degli alpinisti che esso ha soccorso dal 2010 in poi, mostrando anche il codice del cane.

Soluzione: si risolve con un banale join tra le relazioni Soccorso, Alpinista e SanBernardo, al quale si aggiunge una selezione su sesso e anno

```
select B.codice, A.nome, A.annoascita, A.cittanascita
from Soccorso S join Alpinista A on S.codalpinista = A.codice
     join SanBernardo B on S.codsanbernardo = B.codice
where B.sesso = "F" and S.anno >= 2010
```

Problema 3 - Query 2

Testo: Calcolare codice, nome e città di nascita di ogni alpinista che è stato soccorso almeno una volta e che è stato soccorso sempre da cani che avevano lo stesso sesso dell'alpinista.

Soluzione: si può risolvere usando i complemento dell'insieme degli alpinisti che sono stati soccorsi sempre da cani che avevano lo stesso sesso di tali alpinisti. Tale complemento è semplicemente formato da ogni alpinista che è stato soccorso da almeno un cane di sesso diverso dal proprio.

```
select codice, nome, cittanascita
from Alpinista A
where codice in (select codalpinista from Soccorso) and
        codice not in (select S.codalpinista
                       from Soccorso S, SanBernardo B
                       where S.codalpinista = B.codice and A.sesso <> B.sesso)
```

Problema 3 - Query 3 (soluzione con una vista)

Testo: Per ogni valore di sesso del cane e per ogni età del cane (età calcolata al momento del soccorso), calcolare il numero medio annuo di soccorsi effettuati dal 2000 in poi da cani di quel sesso e di quell'età, ma solo se il numero di tali soccorsi è maggiore di 15.

Soluzione: si definisce una vista in cui per ogni soccorso si mette insieme il sesso del cane, la sua età (calcolata come differenza tra l'anno del soccorso e l'anno di nascita del cane) e l'anno del soccorso. Si usa poi una opportuna group-by su tale vista.

```
create view Vista(sexcane, etàcane, annodelsoccorso) as
select B.Sesso, S.anno - B.annonascita, S.anno
from SanBernardo B join Soccorso S on B.codice = S.codsanbernardo
where S.anno >= 2000

select T.sexcane, T.etàcane, count(*) div count(distinct T.annodelsoccorso)
from Vista T
group by T.sexcane, T.etàcane
having count(*) > 15
```

Problema 3 - Query 3 (soluzione con vista “in-line”)

Per ogni valore di sesso del cane e per ogni età del cane (età calcolata al momento del soccorso), calcolare il numero medio annuo di soccorsi effettuati dal 2000 in poi da cani di quel sesso e di quell’età, ma solo se il numero di tali soccorsi è maggiore di 15.

Soluzione: la vista descritta nella prima soluzione viene definita “in-line”,con una “select” nella clausola “from”. Come prima, si usa poi una opportuna group-by su tale vista.

```
select T.sexcane, T.etàcane, count(*) / count(distinct T.annodelsoccorso)
from (select B.Sesso as sexcane, S.anno - B.annonascita as etàcane,
                S.anno as annodelsoccorso
      from SanBernardo B join Soccorso S on B.codice = S.codsanbernardo
      where S.anno >= 2000) T
group by T.sexcane, T.etàcane
having count(*) > 15
```

Problema 4 - soluzione

- Un vincolo di integrità è una condizione che si esprime a livello di schema e che si intende debba essere soddisfatta da tutte le istanze della base di dati, perché individua una condizione necessaria per tutte quelle istanze della base di dati che rappresentano situazioni corrette per l'applicazione

- Per esprimere il vincolo che nessun valore compare contemporaneamente in $PROJ_A(R)$ e $PROJ_D(Q)$ si può inserire nella create table di R la seguente clausola check:

  check (A NOT in (select D from Q))

- Per esprimere il vincolo che ogni valore che compare in $PROJ_B(SEL_{C=3}(R))$ compare anche in $PROJ_E(SEL_{F=5}(Q))$, si può inserire nella create table di R la seguente clausola check:

  check (C <> 3 OR B in (select E from Q where F = 5))