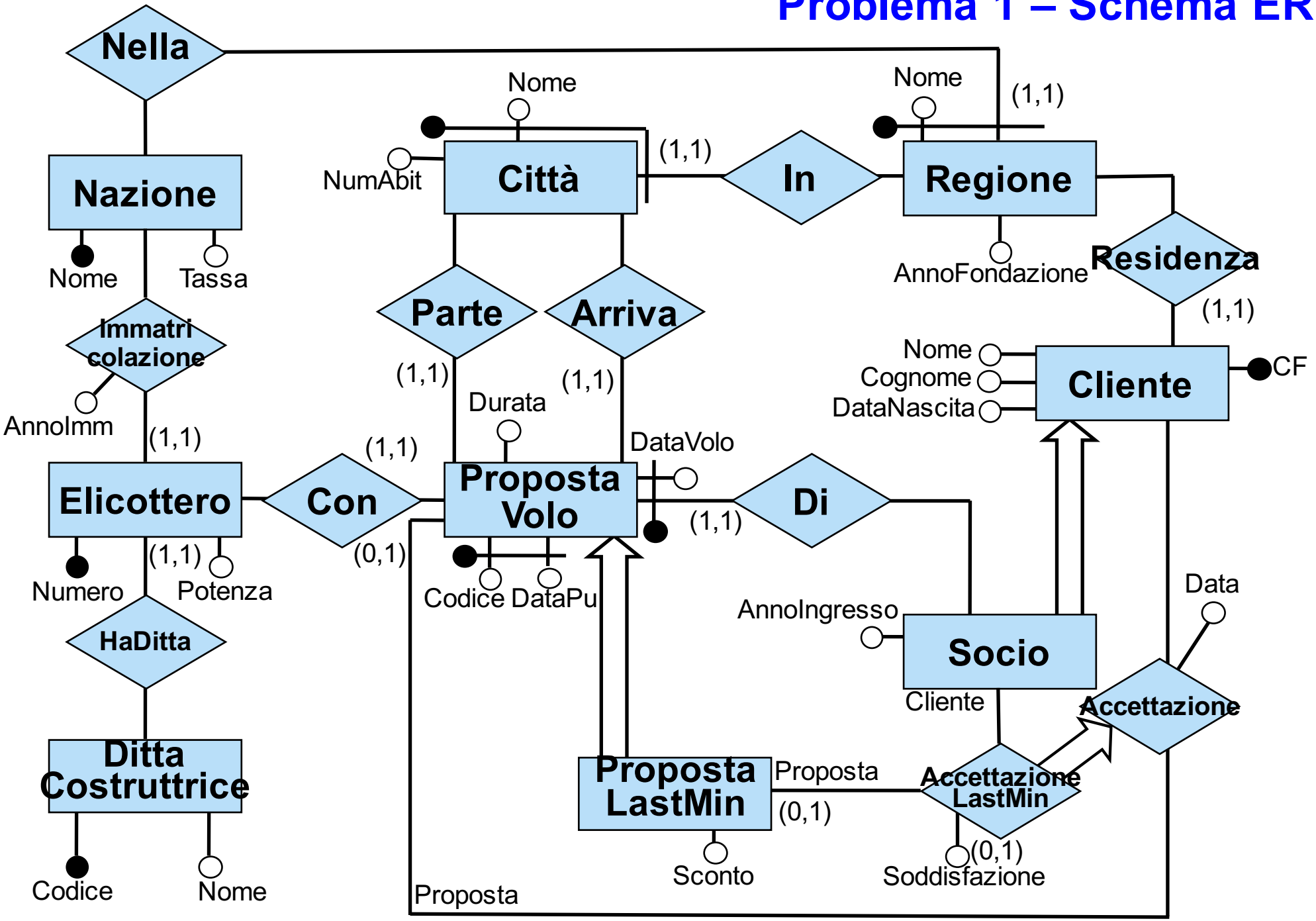
Basi di dati

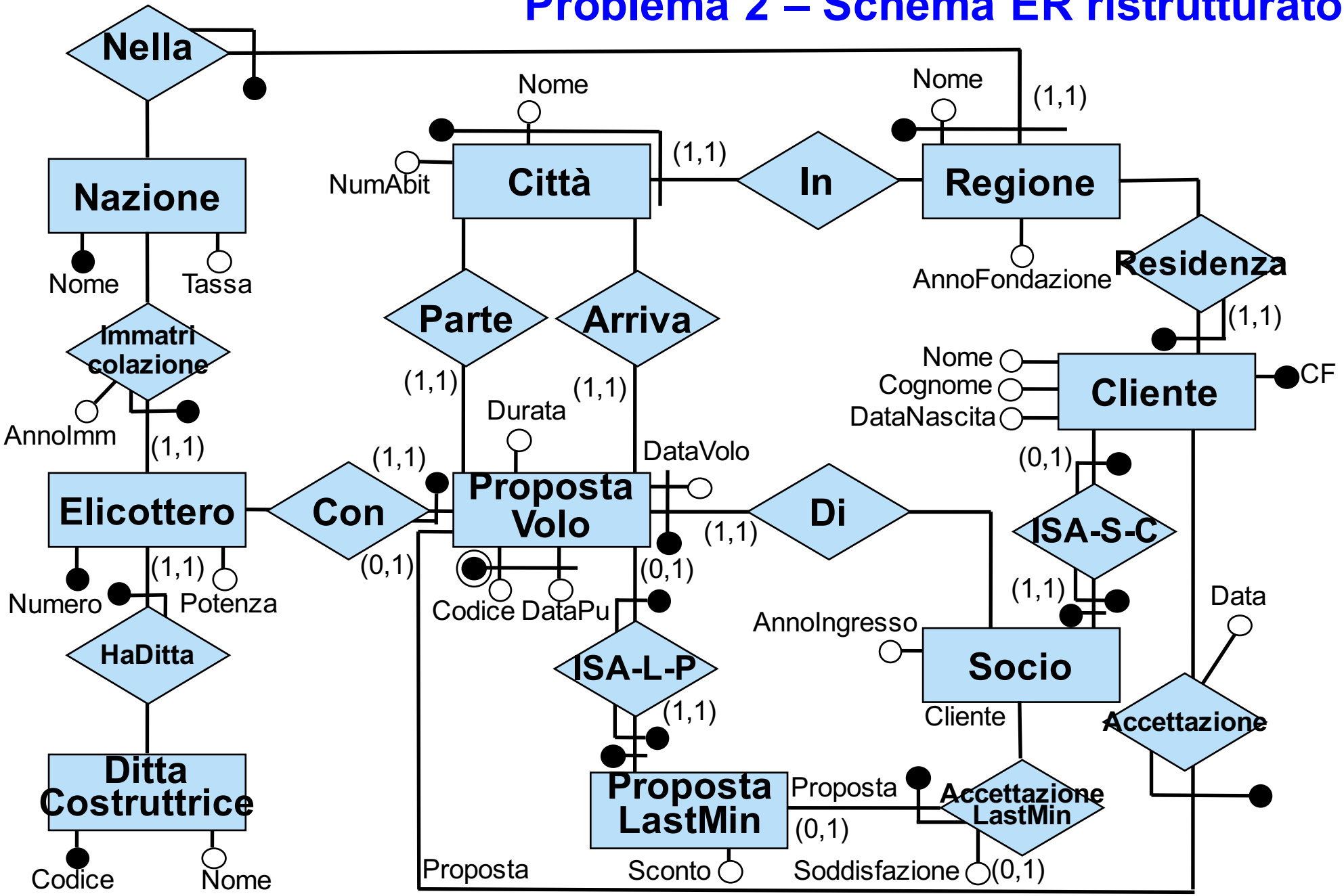
Appello del 8-01-2015
Compito B

Anno Accademico 2014/15

Problema 1 – Schema ER



Problema 2 – Schema ER ristrutturato



Vincolo esterno: Per ogni istanza <PropostaLastMin:p,Cliente:c> di AccettazioneLastMin, se p1 e c1 sono tali che <PropostaLastMin:p,PropostaVolo:p1> è istanza di ISA-L-P e <Socio:c,Cliente:c1> è istanza di ISA-S-C, si ha che l'stanza <PropostaVolo:p1,Cliente:c1> è istanza di Accettazione.

Problema 2 - Schema logico dopo la traduzione diretta

PropostaVolo(Codice, DataPu, DataVolo, Durata)
- foreign key: PropostaVolo[Codice,DataPu] ⊆ Con[CodiceProposta,DataProposta]
- foreign key: PropostaVolo [Codice,DataPu] ⊆ Parte[CodiceProposta,DataProposta]
- foreign key: PropostaVolo [Codice,DataPu] ⊆ Arriva[CodiceProposta,DataProposta]
- foreign key: PropostaVolo [Codice,DataPu] ⊆ Di[CodiceProposta,DataProposta]

Elicottero(Numero, Potenza)
- foreign key: Elicottero[Numero] ⊆ Immatricolazione[Elicottero]
- foreign key: Elicottero[Numero] ⊆ HaDitta[Elicottero]

Con(CodiceProposta, DataProposta, Elicottero)
- foreign key: Con[CodiceProposta,DataProposta] ⊆ PropostaVolo[Codice,DataPu]
- foreign key: Con[Elicottero] ⊆ Elicottero[Numero]

Parte(CodiceProposta, DataProposta, NomeCittà, RegioneCittà, NazioneCittà)
- foreign key: Da[CodiceOfferta,DataOfferta] ⊆ OffertaPassaggio[Codice,DataPu]
- foreign key: Da[NomeCittà,RegioneCittà,NazioneCittà] ⊆ Città[Nome,Regione,Nazione]

Arriva(CodiceProposta, DataProposta, NomeCittà, RegioneCittà)
- foreign key: Arriva[CodiceProposta,DataProposta] ⊆ PropostaVolo[Codice,DataPu]
- foreign key: Arriva[NomeCittà,RegioneCittà,NazioneCittà] ⊆ Città[Nome,Regione,Nazione]

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

Regione(Nome, Nazione, AnnoFondazione)
- foreign key: Regione[Nazione] ⊆ Nazione[Nome]

Nazione(Nome, Tassa)

DittaCostruttrice(Codice, Nome)

HaDitta(Elicottero, Ditta)
- foreign key: HaDitta[Elicottero] ⊆ Elicottero[Numero]
- foreign key: HaDitta[Ditta] ⊆ DittaCostruttrice[Codice]

Problema 2 - Schema logico dopo la traduzione diretta

Immatricolazione(Elicottero, Nazione, AnnoImm)
foreign key: Immatricolazione[Elicottero] ⊆ Elicottero[Numero]
foreign key: Immatricolazione[Nazione] ⊆ Nazione[Nome]
Cliente(CF, Nome, Cognome, DataNascita)
foreign key: Cliente[CF] ⊆ Residenza[Cliente]
Residenza(Cliente, Regione)
foreign key: Residenza[Cliente] ⊆ Cliente[CF]
foreign key: Residenza[Regione] ⊆ Regione[Nome]
Socio(CF, AnnoIscrizione)
foreign key: Socio[CF] ⊆ Cliente[CF]
Di(CodiceProposta, DataProposta, Socio)
foreign key: Di[CodiceProposta,DataProposta] ⊆ PropostaVolo[Codice,DataPu]
foreign key: Di[Socio] ⊆ Socio[CF]
Accettazione(CodiceProposta, DataProposta, Cliente, Data)
foreign key: Accettazione[CodiceProposta,DataProposta] ⊆ PropostaVolo[Codice,DataPu]
foreign key: Accettazione[Cliente] ⊆ Cliente[CF]
PropostaLastMin(Codice, DataPu, Sconto)
foreign key: PropostaLastMin[Codice,DataPu] ⊆ PropostaVolo[Codice,DataPu]
AccettazioneLastMin(CodiceProposta, DataProposta, Socio, Soddisfazione*)
foreign key: AccettazioneLastMin [CodiceProposta,DataProposta] ⊆ PropostaLastMin[Codice,DataPu]
foreign key: AccettazioneLastMin [Socio] ⊆ Socio[CF]
foreign key: AccettazioneLastMin [CodiceProposta,DataProposta,Socio] ⊆
Accettazione[CodiceProposta,DataProposta,Cliente]

Vincolo esterno: Nell'equi-join tra Di e PropostaVolo con condizione di join (CodiceProposta=Codice AND DataProposta=DataPu]), gli attributi DataVolo e Socio formano una chiave.

Problema 2 - Schema logico dopo la ristrutturazione

- *Indicazione 2*: Accorpamento di PropostaVolo e Con (sfruttando l'accoppiamento forte)
- *Indicazione 3*: Accorpamento di PropostaVolo e Accettazione (sfruttando l'accorpamento debole), considerando il valore nullo/non nullo come un flag, in modo che le tuple di Proposta 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.

PropostaVolo(Codice, DataPu, DataVolo, Durata,Cliente*,DataAccettazione*)
foreign key: PropostaVolo[Codice,DataPu] ⊆ Con[CodiceProposta,DataProposta]
foreign key: PropostaVolo[Codice,DataPu] ⊆ Parte[CodiceProposta,DataProposta]
foreign key: PropostaVolo[Codice,DataPu] ⊆ Arriva[CodiceProposta,DataProposta]
foreign key: PropostaVolo[Codice,DataPu] ⊆ Di[CodiceProposta,DataProposta]
foreign key: PropostaVolo[Cliente] ⊆ Cliente[CF]
vincolo di tupla: (Cliente <> NULL OR DataAccettazione = NULL) AND
(DataAccettazione <> NULL OR Cliente = NULL)

Elicottero(Numero, Potenza, 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]

Parte(CodiceProposta, DataProposta, NomeCittà, RegioneCittà, NazioneCittà)
foreign key: Parte[CodiceProposta,DataProposta] ⊆ PropostaVolo[Codice,DataPu]
foreign key: Parte[NomeCittà,RegioneCittà,NazioneCittà] ⊆ Città[Nome,Regione,Nazione]

Arriva(CodiceProposta, DataProposta,NomeCittà, RegioneCittà, NazioneCittà)
foreign key: Arriva[CodiceProposta,DataProposta] ⊆ Proposta[Codice,DataPu]
foreign key: Arriva[NomeCittà,RegioneCittà,NazioneCittà] ⊆ Città[Nome,Regione,Nazione]

Problema 2 - Schema logico dopo la ristrutturazione

Città(Nome, Regione, Nazione, NumAbit)
foreign key: Città[Regione,Nazione] ⊆ Regione[Nome,Nazione]
Regione(Nome, Nazione, AnnoFondazione)
foreign key: Regione[Nazione] ⊆ Nazione[Nome]
Nazione(Nome, Tassa)
DittaCostruttrice(Codice, Nome)
HaDitta(Elicottero,Ditta)
foreign key: HaDitta[Elicottero] ⊆ Elicottero[Numero]
foreign key: HaDitta[Ditta] ⊆ DittaCostruttrice[Codice]
Cliente(CF, Nome, Cognome, DataNascita)
foreign key: Persona[CF] ⊆ Nascita[Persona]
Residenza(Cliente, Regione)
foreign key: Residente[Cliente] ⊆ Cliente[CF]
foreign key: Residenza[Regione] ⊆ Regione[Nome]
Socio(CF, AnnoIscrizione)
foreign key: Socio[CF] ⊆ Cliente[CF]
Di(CodiceProposta, DataProposta, Socio)
foreign key: Di[CodiceProposta,DataProposta] ⊆ PropostaVolo[Codice,DataPu]
foreign key: Di[Socio] ⊆ Socio[CF]
PropostaLastMin(Codice, DataPu, Sconto)
foreign key: OffertaSpeciale[Codice,DataPu] ⊆ OffertaPassaggio[Codice,DataPu]
AccettazioneLasMin(CodiceProposta, DataProposta, Cliente, Voto*)
foreign key: AccettazioneLasMin [CodiceProposta,DataProposta] ⊆ PropostaLastMin[Codice,DataPu]
foreign key: AccettazioneLasMin [Cliente] ⊆ Socio[CF]
foreign key: AccettazioneLasMin [CodiceProposta,DataProposta,Cliente] ⊆
Accettazione[CodiceProposta,DataProposta,Cliente]

Vincolo esterno: Nell'equi-join tra Di e PropostaVolo con condizione di join (CodiceProposta=Codice AND DataProposta=DataPu]), gli attributi DataVolo e Socio formano una chiave.

Problema 3 - Query 1

Testo: Per ogni medico femmina, calcolare il nome, l'anno di nascita e la città di nascita degli sciatori dei quali ha gestito i ricoveri dal 2010 in poi, mostrando anche il codice del medico.

Soluzione: si risolve con un banale join tra le relazioni Ricovero, Sciatore e Medico, al quale si aggiunge una selezione su sesso e anno

```
select M.codice, S.nome, S.annoascita, S.cittanascita
from Ricovero R join Sciatore S on R.codsciatore = S.codice
        join Medico M on R.codmedico = M.codice
where M.sesso = "F" and R.anno >= 2010
```

Problema 3 - Query 2

Testo: Calcolare codice, nome e città di nascita di ogni sciatore che è stato ricoverato almeno una volta e che è stato ricoverato sempre da medici che avevano lo stesso sesso dello sciatore.

Soluzione: si può risolvere usando i complemento dell'insieme degli sciatori che sono stati ricoverati sempre da medici che avevano lo stesso sesso di tali sciatori. Tale complemento è semplicemente formato da ogni sciatore il cui ricovero è stato gestito a almeno un medico di sesso diverso dal proprio.

```
select codice, nome, cittanascita
from Sciatore S
where codice in (select codsciatore from Ricovero) and
        codice not in (select R.codsciatore
                       from Ricovero R, Medico M
                       where R.codmedico = M.codice and M.sesso <> S.sesso)
```

Problema 3 - Query 3 (soluzione con una vista)

Testo: Per ogni valore di sesso del medico e per ogni valore di età del medico (età calcolata al momento del ricovero), calcolare il numero medio annuo di ricoveri gestiti dal 2000 in poi da medici di quel sesso e a quell'età, ma solo se il numero di tali ricoveri è maggiore di 15.

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

```
create view Vista(sexmedico, etàmedico, annodelricovero) as
select M.Sesso, R.anno - M.annonascita, R.anno
from Medico M join Ricovero R on M.codice = R.codmedico
where R.anno >= 2000

select T.sexmedico, T.etàmedico, count(*) div count(distinct T.annodelricovero)
from Vista T
group by T.sexmedico, T.etàmedico
having count(*) > 15
```

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

Testo: Per ogni valore di sesso del medico e per ogni valore di età del medico (età calcolata al momento del ricovero), calcolare il numero medio annuo di ricoveri gestiti dal 2000 in poi da medici di quel sesso e a quell’età, ma solo se il numero di tali ricoveri 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.sexmedico, T.etàmedico, count(*) / count(distinct T.annodelricovero)
from (select M.Sesso as sexmedico, R.anno - M.annonascita as etàmedico,
             R.anno as annodelricovero
      from Medico M join Ricovero R on M.codice = R.codmedico
      where R.anno >= 2000) T
group by T.sexmedico, T.etàmedico
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 ogni valore che compare in $PROJ_{A2}(SEL_{A3=3}(T1))$ compare anche in $PROJ_{B1}(SEL_{B3=5}(T2))$ si può inserire nella create table di T1 la seguente clausola check:

  check (A3 <> 3 OR A2 in (select B1 from T2 where B3 = 5))
- Per esprimere il vincolo che se un valore compare nella stessa tupla di T1 sia nell'attributo A1 sia nell'attributo A2, allora compare in una qualunque tupla e in un qualunque attributo nella relazione T2 si può inserire nella create table di T1 la seguente clausola check:

```
check (A1 <> A2 OR A1 in (select B1 from T2
                          union
                          select B2 from T2
                          union
                          select B3 from T2))
```