

Costruire il DB “Compagnia telefonica”

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)

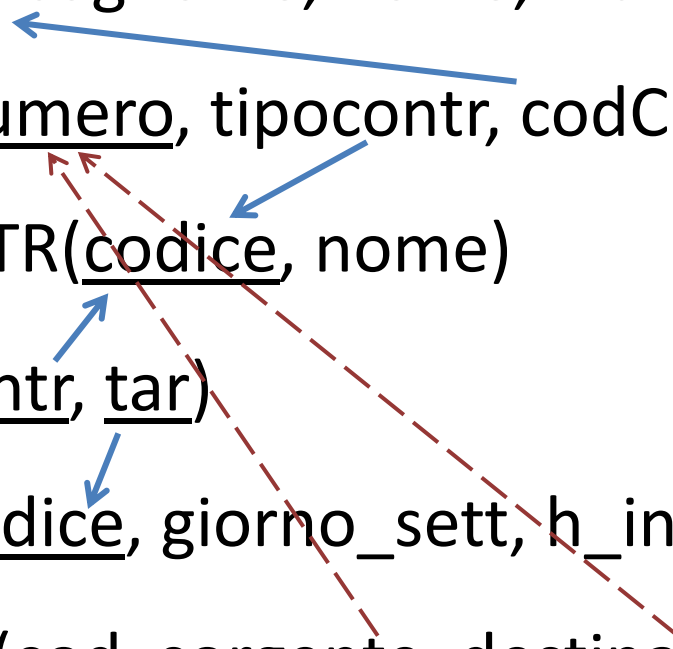
UTENZA(numero, tipocontr, codCliente)

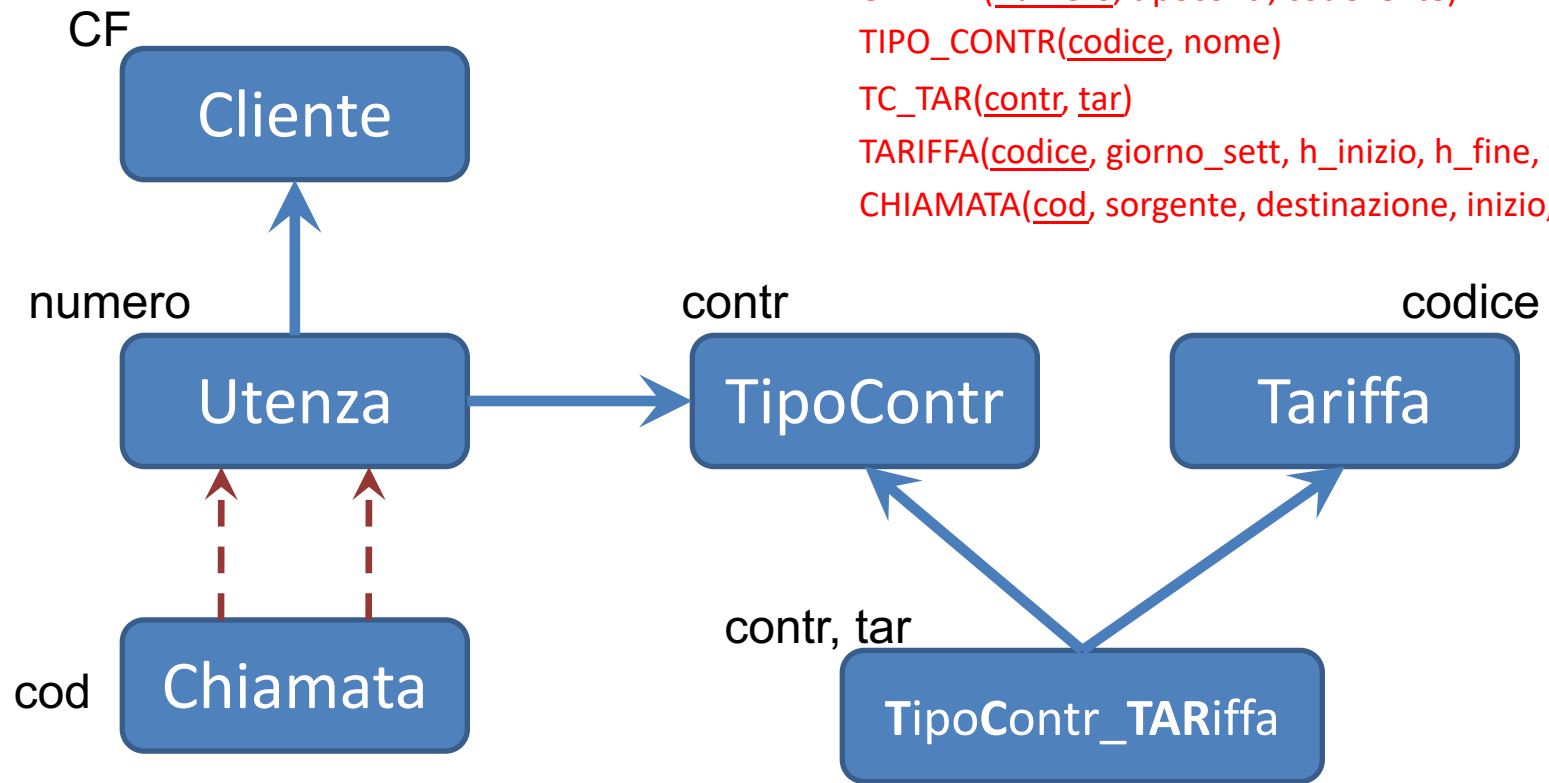
TIPO_CONTR(codice, nome)

TC_TAR(contr, tar)

TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)

CHIAMATA(cod, sorgente, destinazione, inizio, fine)





CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)
UTENZA(numero, tipocontr, codCliente)
TIPO_CONTR(codice, nome)
TC_TAR(contr, tar)
TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)
CHIAMATA(cod, sorgente, destinazione, inizio, fine)

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)

```
CREATE TABLE cliente (  
    cf char(16) primary key,  
    cognome varchar(50) not null,  
    nome varchar(50) not null,  
    indirizzo varchar(255) ,  
    città varchar(30) ,  
    domic BOOLEAN,  
    iban char(25)  
)
```

UTENZA(numero, tipocontr, codCliente)

```
CREATE TABLE utenza(  
    numero char(12) primary key,  
    tipocontr integer  
                references tipo_contr(codice)  
                on delete no action  
                on update cascade  
    codCliente char(16) references cliente(cf)  
                on delete no action  
                on update cascade  
)
```

TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)

```
CREATE DOMAIN DAYS-DOMAIN AS char(3)
CHECK (VALUE IN ( 'lun' , 'mar' , ... , 'dom' ) ) ;

CREATE TABLE tariffa(
    codice integer primary key,
    giorno_sett DAYS-DOMAIN,
    h_inizio time,
    h_fine time,
    flat BOOLEAN,
    cifra decimal
)
```

TC_TAR(contr, tar)

```
CREATE TABLE TC_TAR(  
  contr integer references tipo_contr(codice)  
    on delete set null  
    on update cascade,  
  tar integer references tariffa(codice)  
    on delete set null  
    on update cascade,  
  PRIMARY KEY (contr, tar))
```

Non è possibile mettere a
null un valore di chiave!
“cascade” o “no action”

DML

Costruire il DB “Compagnia telefonica”

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)

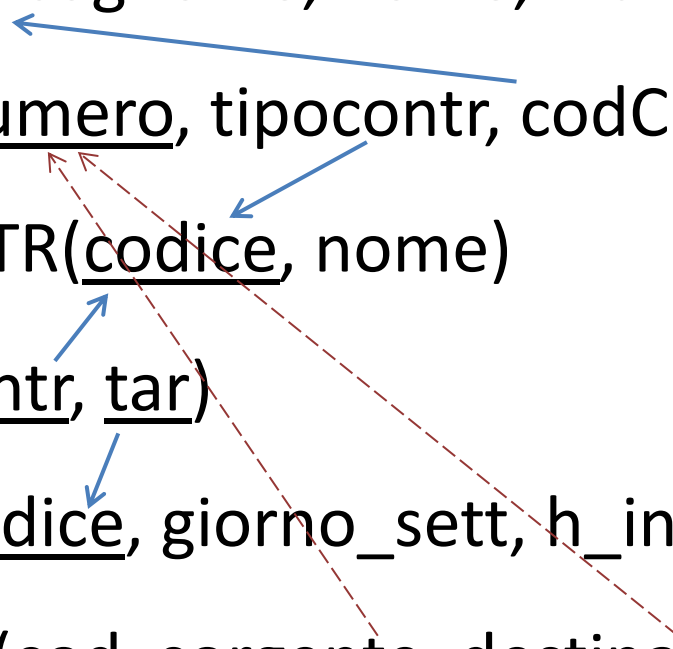
UTENZA(numero, tipocontr, codCliente)

TIPO_CONTR(codice, nome)

TC_TAR(contr, tar)

TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)

CHIAMATA(cod, sorgente, destinazione, inizio, fine)



1) Tutti i clienti di Milano

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)

UTENZA(numero, tipocontr, codCliente)

TIPO_CONTR(codice, nome)

TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)

TC_TAR(contr, tar)

CHIAMATA(cod, sorgente, destinazione, inizio, fine)

```
select *  
from cliente  
where citta = 'Milano'
```

2) *Elenco telefonico di Milano*

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)

UTENZA(numero, tipocontr, codCliente)

TIPO_CONTR(codice, nome)

TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)

TC_TAR(contr, tar)

CHIAMATA(cod, sorgente, destinazione, inizio, fine)

```
select cognome, nome, indirizzo,  
       citta, numero  
from cliente, utenza  
where cf = codCliente  
      and citta = 'Milano'  
order by cognome, nome
```

2) *Elenco telefonico di Milano* *(sintassi alternativa)*

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)

UTENZA(numero, tipocontr, codCliente)

TIPO_CONTR(codice, nome)

TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)

TC_TAR(contr, tar)

CHIAMATA(cod, sorgente, destinazione, inizio, fine)

```
select cognome, nome, indirizzo,  
       citta, numero  
from cliente [inner] join utenza  
       on cf = codCliente  
where citta = 'Milano'  
order by cognome, nome
```

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)

UTENZA(numero, tipocontr, codCliente)

TIPO_CONTR(codice, nome)

TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)

TC_TAR(contr, tar)

CHIAMATA(cod, sorgente, destinazione, inizio, fine)

3) *Il tipo di contratto di ogni cliente*

```
select c.cf, c.cognome, c.nome, t.nome
from (cliente as c join utenza as u
      on c.cf = u.codCliente)
      join tipo_contr as t
      on u.tipocontr = t.codice
```

*4) Dettagli della chiamata
più recente
(in tutto il sistema)*

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)
UTENZA(numero, tipocontr, codCliente)
TIPO_CONTR(codice, nome)
TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)
TC_TAR(contr, tar)
CHIAMATA(cod, sorgente, destinazione, inizio, fine)

```
select sorgente, dest, inizio, fine
from chiamata
where inizio = ( select max(inizio)
                  from chiamata )
```

*4) Dettagli della chiamata
più recente
(in tutto il sistema)*

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)
UTENZA(numero, tipocontr, codCliente)
TIPO_CONTR(codice, nome)
TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)
TC_TAR(contr, tar)
CHIAMATA(cod, sorgente, destinazione, inizio, fine)

Errori tipici:

**select *
from chiamata
where max(inizio)**

**select max(inizio)
from chiamata**

**select Sorg, Dest, in, fin, max(inizio)
from chiamata**

5) *Dettagli della chiamata
più recente
(per ogni numero)*

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)
UTENZA(numero, tipocontr, codCliente)
TIPO_CONTR(codice, nome)
TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)
TC_TAR(contr, tar)
CHIAMATA(cod, sorgente, destinazione, inizio, fine)

```
select cf, cognome, nome, chiamata.*  
from (cliente join utenza u   
      on cf = codCliente) join chiamata  
      on u.numero = sorgente  
where inizio = ANY/ALL( select max(inizio)  
                        from chiamata  
                        where sorgente = u.numero)
```

*BINDING: Variabile
dichiarata fuori e
utilizzata nella
query interna*

78

6) *Quando è iniziata la chiamata
più recente
(partita da) ogni numero?*

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)
UTENZA(numero, tipocontr, codCliente)
TIPO_CONTR(codice, nome)
TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)
TC_TAR(contr, tar)
CHIAMATA(cod, sorgente, destinazione, inizio, fine)

```
select numero, max(inizio)
from utenza join chiamata
      on numero = sorgente
group by numero
```

```
select sorgente, max(inizio)
from chiamata
group by sorgente
```

Alternativamente, è possibile
raggruppare su sorgente
→ in effetti è sufficiente la
tabella *chiamata* !

QUAL E' LA DIFFERENZA ?

ATTENZIONE

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)

UTENZA(numero, tipocontr, codCliente)

TIPO_CONTR(codice, nome)

TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)

TC_TAR(contr, tar)

CHIAMATA(cod, sorgente, destinazione, inizio, fine)

**Con questo approccio, però, non possiamo
estrarre i dettagli della chiamata**

~~select **chiamata.***, max(inizio)
from chiamata
group by **sorgente**~~

Perché ???

7) *Per ogni numero, la chiamata
più recente
(tra quelle dopo il 01/01/2009)*

```
select sorgente, max(inizio)
from chiamata
where inizio > 01/01/2009
group by sorgente
```

```
select sorgente, max(inizio)
from chiamata
group by sorgente
having max(inizio) > 01/01/2009
```

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)

UTENZA(numero, tipocontr, codCliente)

TIPO_CONTR(codice, nome)

TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)

TC_TAR(contr, tar)

CHIAMATA(cod, sorgente, destinazione, inizio, fine)

Versione 1: scarta subito le vecchie chiamate e raggruppa solo quelle recenti.

Versione 2: raggruppa tutte le chiamate, **poi** scarta i gruppi in cui la chiamata più recente è troppo vecchia.

In questo caso, sono equivalenti (*anche se c'è una differenza... quale?*)

8) *Numero di linee telefoniche di
ogni utente
(purché siano più di 2)*

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)
UTENZA(numero, tipocontr, codCliente)
TIPO_CONTR(codice, nome)
TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)
TC_TAR(contr, tar)
CHIAMATA(cod, sorgente, destinazione, inizio, fine)

```
select codCliente, count(*)  
from utenza  
group by codCliente  
having count(*) > 2
```

...ora dovrebbe essere chiara la differenza tra clausole

WHERE (valutate *prima* del raggruppamento, per ogni tupla) e clausole

HAVING (valutate *dopo* il raggruppamento, per ogni gruppo)

9) *Clienti che hanno telefonato a linee intestate a qualcuno che non risiede nella stessa città, e quindi sono probabilmente iniziatori di telefonate interurbane*

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)

UTENZA(numero, tipocontr, codCliente)

TIPO_CONTR(codice, nome)

TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)

TC_TAR(contr, tar)

CHIAMATA(cod, sorgente, destinazione, inizio, fine)

```
select distinct c1.*
from cliente c1
  join utenza u1 on c1.cf = u1.codCliente
  join chiamata ch on u1.numero = ch.sorgente
  join utenza u2 on ch.destinazione = u2.numero
  join cliente c2 on u2.codCliente = c2.cf
where c1.citta <> c2.citta
```

*10) Numero di chiamate tra utenze
con lo stesso tipo di contratto nel
giorno 08/11/2006*

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)

UTENZA(numero, tipocontr, codCliente)

TIPO_CONTR(codice, nome)

TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)

TC_TAR(contr, tar)

CHIAMATA(cod, sorgente, destinazione, inizio, fine)

```
select count(*)  
from utenza u1  
      join chiamata on u1.numero = sorgente  
      join utenza u2 on destinazione = u2.numero  
where inizio between '08/11/2006 00:00:00'  
              and '08/11/2006 23:59:59'  
and u1.tipocontr = u2.tipocontr
```

11) Numero di utenze che hanno fatto chiamate verso utenze con lo stesso tipo di contratto nel giorno 08/11/2006

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)
UTENZA(numero, tipocontr, codCliente)
TIPO_CONTR(codice, nome)
TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)
TC_TAR(contr, tar)
CHIAMATA(cod, sorgente, destinazione, inizio, fine)

```
select count(distinct sorgente)
from utenza u1
      join chiamata on u1.numero = sorgente
      join utenza u2 on destinazione = u2.numero
where inizio between '08/11/2006 00:00:00'
                  and '08/11/2006 23:59:59'
and u1.tipocontr = u2.tipocontr
```

12) I contratti le cui tariffe non superino il 5cent/sec

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)

UTENZA(numero, tipocontr, codCliente)

TIPO_CONTR(codice, nome)

TARIFFA(codice, giorno_sett, h_inizio, h_fine, flat, cifra)

TC_TAR(contr, tar)

CHIAMATA(cod, sorgente, destinazione, inizio, fine)

~~```
select nome
from t_contr
where codice in (
 select tc_tar.contr
 from tc_tar join tariffa
 on tariffa.codice = tc_tar.tar
 where tariffa.cifra <= 5)
```~~

**No !!!**

Questa soluzione verifica che  
**ALMENO UNA** tariffa prevista  
dal contratto abbia un costo  
**inferiore a 5 cent/sec, NON**  
che il tipo di contratto non costi  
**MAI** più di così

*12) I contratti le cui tariffe non  
superino il 5cent/sec*

```
select nome
from t_contr
where codice not in (
 select tc_tar.contr
 from tc_tar join tariffa
 on tariffa.codice = tc_tar.tar
where tariffa.cifra > 5)
```

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)

UTENZA(numero, tipocontr, codCliente)

TIPO\_CONTR(codice, nome)

TARIFFA(codice, giorno\_sett, h\_inizio, h\_fine, flat, cifra)

TC\_TAR(contr, tar)

CHIAMATA(cod, sorgente, destinazione, inizio, fine)

Questa soluzione **esclude**  
tutte le tipologie che  
abbiano ALMENO UNA  
tariffazione **sopra** la soglia



*12) I contratti le cui tariffe non  
superino il 5cent/sec*

CLIENTE(cf, cognome, nome, indirizzo, città, domic, iban)  
UTENZA(numero, tipocontr, codCliente)  
TIPO\_CONTR(codice, nome)  
TARIFFA(codice, giorno\_sett, h\_inizio, h\_fine, flat, cifra)  
TC\_TAR(contr, tar)  
CHIAMATA(cod, sorgente, destinazione, inizio, fine)

```
select nome
from t_contr
where codice in (
 select tc_tar.contr
 from tc_tar join tariffa
 on tariffa.codice = tc_tar.tar
 group by tc_tar.contr
 having max(tariffa.cifra) <= 5)
```

# Forniture prodotti

Fornitori ( CodiceForn, Nome, Indirizzo, Città)

Prodotti ( CodiceProd, Nome, Marca, Modello)

Catalogo ( CodiceForn, CodiceProd, Costo)

*13) I codici di tutti i prodotti distribuiti da almeno due fornitori*

Fornitori ( CodiceForn, Nome, Indirizzo, Città)  
Prodotti ( CodiceProd, Nome, Marca, Modello)  
Catalogo ( CodiceForn, CodiceProd, Costo)

```
select distinct C.CodiceProd
from Catalogo AS C, Catalogo AS C1
where C.CodiceProd = C1.CodiceProd
 and C.CodiceForn > C1.CodiceForn
```

Corrisponde all'espressione algebrica

$$\Pi_{CP} ( \text{Catalogo} \mid \! \! \! \rangle \! \! \! \langle_{CP=CP \wedge CF > CF} \text{Catalogo} )$$

o, più precisamente:

$$\Pi_{CP} ( \text{Catalogo} \mid \! \! \! \rangle \! \! \! \langle_{CP=cp1 \wedge CF > cf1} ( \rho_{CP \leftarrow cp1, CF \leftarrow cf1} \text{Catalogo} ) )$$

90

**< ESERCIZIARIO >**

*13) I codici di tutti i prodotti distribuiti da  
almeno due fornitori*

Fornitori ( CodiceForn, Nome, Indirizzo, Città)  
Prodotti ( CodiceProd, Nome, Marca, Modello)  
Catalogo ( CodiceForn, CodiceProd, Costo)

La chiave di Catalogo è data da <CodiceForn,CodiceProd>  
e in SQL possiamo predicare sui gruppi:

```
select CodiceProd
from Catalogo
group by CodiceProd
having count(*) > 1
```

**MOLTO PIÙ EFFICIENTE**

*Non solo agisce su una sola  
tabella, ma tipicamente  
riesce a sfruttare un **indice**  
definito sulla chiave per  
raggruppare rapidamente*

**< ESERCIZIARIO >**

Fornitori ( CodiceForn, Nome, Indirizzo, Città)  
Prodotti ( CodiceProd, Nome, Marca, Modello)  
Catalogo ( CodiceForn, CodiceProd, Costo)

*14) Quali prodotti sono forniti da più di un fornitore, ma allo stesso costo da parte di tutti?*

```
select CodiceProd
from Catalogo
group by CodiceProd
having count(*) > 1 and
 count(distinct Costo) = 1
```

*Posso sfruttare il raggruppamento della query precedente e ricavare il risultato in base alle proprietà aggregate dei gruppi risultanti*

15) *Di ogni prodotto, calcolare il costo **medio**  
di fornitura per ciascuna città*

```
select codiceprod, città, avg(costo) as media
from catalogo c, fornitori f
where c.codiceforn = f.codiceforn
group by città, codiceprod
```


N.B. L'ordine degli attributi della GROUP BY non è mai importante (a differenza dell'ORDER-BY)!

< **ESERCIZIARIO** >

16) *Nomi dei fornitori “universali”, cioè che distribuiscono tutti i prodotti in catalogo*

Fornitori ( CodiceForn, Nome, Indirizzo, Città)  
Prodotti ( CodiceProd, Nome, Marca, Modello)  
Catalogo ( CodiceForn, CodiceProd, Costo)

```
select CodiceForn, Nome
from Fornitori F
where not exists
(select *
 from Prodotti P
 where not exists
 (select *
 from Catalogo C1
 where C1.CodiceForn = F.CodiceForn
 and C1.CodiceProd = P.CodiceProd))
```



94

< **ESERCIZIARIO** >

16) *Nomi dei fornitori “universali”, cioè che distribuiscono tutti i prodotti in catalogo*

Fornitori ( CodiceForn, Nome, Indirizzo, Città)  
Prodotti ( CodiceProd, Nome, Marca, Modello)  
Catalogo ( CodiceForn, CodiceProd, Costo)

```
select CodiceForn, Nome
from Fornitori
where CodiceForn not in
(select CodiceForn
 from Prodotti, Fornitori
 where (CodiceProd, CodiceForn) not in
 (select CodiceProd, CodiceForn
 from Catalogo))
```

Versione SENZA “NOT EXISTS”  
e variabili:  
fa uso solo dei NOT IN



16) *Nomi dei fornitori “universali”, cioè che distribuiscono tutti i prodotti in catalogo*

Fornitori ( CodiceForn, Nome, Indirizzo, Città)  
Prodotti ( CodiceProd, Nome, Marca, Modello)  
Catalogo ( CodiceForn, CodiceProd, Costo)

Versione con aggregati

```
select Nome
from Fornitori F join Catalogo C
 on F.CodiceForn=C.CodiceForn
group by F.CodiceForn, Nome
having count(*) = (select count(*)
 from Prodotti)
```

< **ESERCIZIARIO** >

**REGISTA ( Nome, DataNascita, Nazionalità )**

**ATTORE ( Nome, DataNascita, Nazionalità )**

**INTERPRETA ( Attore, Film, Personaggio )**

**FILM ( Titolo, NomeRegista, Anno)**

**PROIEZIONE ( NomeCin, CittàCin, TitoloFilm )**

**CINEMA ( Città, NomeCinema, #Sale, #Posti )**

*Modifichiamo **lo schema**, aggiungendo un attributo 'Tipo' alla tabella Film*

```
alter table FILM
add column Tipo varchar(20)
 default 'Normale'
```

**< ESERCIZIARIO >**

*Diamo poi al nuovo attributo il valore “Flop” se il film è attualmente in proiezione in meno di 10 cinema*

```
update FILM
 set Tipo = "Flop"
 where 10 > (select count(*)
 from PROIEZIONE
 where TitoloFilm = Titolo)
```

**< ESERCIZIARIO >**

50) *Selezionare le Nazionalità dei registi che hanno diretto qualche film nel 1992 **ma non** hanno diretto **alcun** film nel 1993*

```
select distinct Nazionalità
from REGISTA
where Nome in (select NomeRegista
 from FILM where Anno='1992')
and Nome not in (select NomeRegista
 from FILM where Anno='1993')
```

100

< **ESERCIZIARIO** >

*In alternativa si può usare **except** (a patto di discriminare in base alla chiave) ma assolutamente NON un join e la condizione*

~~where Anno = 1992 and Anno <> 1993~~

*perché la clausola where agisce a livello di TUPLA*

select Nazionalità

from REGISTA

where **Nome** in

( select **NomeRegista** from FILM where Anno = 1992

**except**

select **NomeRegista** from FILM where Anno = 1993 )

101

< **ESERCIZIARIO** >

*NON si può usare la EXCEPT direttamente se nella target list non è incluso l'attributo discriminante per l'esclusione*

~~select Nazionalità  
from FILM join REGISTA on NomeRegista=Nome  
where Anno = 1992~~

**except**

~~select Nazionalità  
from FILM join REGISTA on NomeRegista=Nome  
where Anno = 1993~~

*< ESERCIZIARIO >*

~~select Nazionalità  
from FILM join REGISTA on NomeRegista=Nome  
where Anno = 1992~~

**except**

~~select Nazionalità  
from FILM join REGISTA on NomeRegista=Nome  
where Anno = 1993~~

*Un solo regista di nazionalità X che abbia girato un film nel '93 farebbe sparire dalla soluzione il valore X, anche se molti altri suoi connazionali fossero tali da soddisfare l'interrogazione. Attenzione: in SQL gli operatori insiemistici eliminano i duplicati (come se davanti a Nazionalità ci fosse «distinct»)*

# < ESERCIZIARIO >



51) Individuare le date di nascita dei registi che hanno diretto film in proiezione sia a **Torino** sia a **Milano**

```
select distinct DataNascita
from REGISTA join FILM on Nome=NomeRegista
where Titolo in (select NomeFilm
 from PROIEZIONE
 where CittàCin = 'Milano')
and Titolo in (select NomeFilm
 from PROIEZIONE
 where CittàCin = 'Torino')
```

104

< **ESERCIZIARIO** >

In alternativa si può usare **intersect** (sempre utilizzando la chiave). Qui è ancora più evidente che

~~where Città="Torino" and Città="Milano"~~

è un predicato palesemente auto-contraddittorio

< ***ESERCIZIARIO*** >

```
select DataNascita
from REGISTA
where Nome in
(select NomeRegista
 from FILM join PROIEZIONE on Titolo = TitoloFilm
 where Città = 'Milano'
 intersect
 select NomeRegista
 from FILM join PROIEZIONE on Titolo = TitoloFilm
 where Città = 'Torino')
```

< ***ESERCIZIARIO*** >

*NON si può usare una **intersect** diretta*

~~( select DataNascita  
from (FILM join REGISTA on Titolo=Nome)  
join PROIEZIONE on Titolo=TitoloFilm  
where Città = 'Milano' )~~

~~**intersect**~~

~~( select DataNascita  
from (FILM join REGISTA on Titolo=Nome)  
join PROIEZIONE on Titolo=TitoloFilm  
where Città = 'Torino' )~~

*< ESERCIZIARIO >*

~~( select DataNascita  
from (FILM join REGISTA on Titolo=Nome)  
join PROIEZIONE on Titolo=TitoloFilm  
where Città = 'Milano' )~~

**intersect**

~~( select DataNascita  
from (FILM join REGISTA on Titolo=Nome)  
join PROIEZIONE on Titolo=TitoloFilm  
where Città = 'Torino' )~~

Due registi *nati lo stesso giorno*  
tali che i film dell'uno siano stati  
proiettati *solo* a Torino e quelli  
dell'altro *solo* a Milano  
contribuirebbero erroneamente  
alla soluzione con la loro data di  
nascita

< **ESERCIZIARIO** >

52) *Nomi dei registi che hanno diretto nel 1993 **più film** di quanti ne avevano diretti nel 1992*


```
select NomeRegista
from FILM as F
where Anno='1993'
group by NomeRegista
having count(*) >
 (select count(*)
 from FILM as F1
 where F1.NomeRegista=F.NomeRegista
 and Anno='1992')
```

109

< ***ESERCIZIARIO*** >

## E INVERTENDO GLI ANNI E IL PREDICATO?

```
select NomeRegista
from FILM as F
where Anno='1992'
group by NomeRegista
having count(*) <
(
 select count(*)
 from FILM as F1
 where F1.NomeRegista=F.NomeRegista
 and Anno='1993')
```



È errata: dimentica i registi  
che non hanno diretto  
ALCUN film nel '92

< **ESERCIZIARIO** >

*Oppure si può usare una vista intermedia*

Create View NumPerAnno (Nom, Ann, Num) AS

select NomeRegista, Anno, count(\*)

from FILM

**group by NomeRegista, Anno**

select Nom as NomeRegistaCercato

from NumPerAnno **N1**

where Ann = 1993 and

Nom **not in** ( select Nom from NumPerAnno **N2**

where N2.Ann = 1992 and

N1.Num <= N2.Num )

*Un self join "semplice" su  
NumPerAnno perderebbe  
chi non ha girato film nel 92  
– si potrebbe fare un self-  
outer-join? Come??*

111

**< ESERCIZIARIO >**



*Alternativa con **due** viste intermedie (e senza not-in)*

|                                                                   |             |
|-------------------------------------------------------------------|-------------|
| Create view Num93 (Nom, Num) AS                                   | ....Num92   |
| select NomeRegista, count(*)                                      | ....        |
| from FILM                                                         |             |
| where Anno = 1993                                                 | .... = 1992 |
| <b>group by NomeRegista, Anno</b>                                 |             |
| select Nom as NomeRegistaCercato                                  |             |
| from Num93 <b>N3 left join</b> Num92 <b>N2 on N3.Nom = N2.Nom</b> |             |
| where N3.Num > N2.Num                                             |             |
| <b>or N2.Num is null</b>                                          |             |

53) Film proiettati nel maggior numero di cinema di Milano

```
select TitoloFilm, count(*) as NumCin
```

```
from PROIEZIONE
```

```
where CittàCin = 'Milano'
```

```
group by TitoloFilm
```

```
having count(*) >= ALL (select count(*)
```

```
from PROIEZIONE
```

```
where CittàCin = 'Milano'
```

```
group by TitoloFilm)
```

*NumCin non è richiesto  
dalla specifica, ma  
migliora la leggibilità*

113

< **ESERCIZIARIO** >

*Oppure si può usare una vista intermedia*

```
Create View ProiezMilano (Titolo, Num) AS
 select TitoloFilm, count(*)
 from PROIEZIONE
 where CittàCin = 'Milano'
 group by TitoloFilm
```

```
select Titolo
from ProiezMilano
where Num = (select max(Num)
 from ProiezMilano)
```

114

**< ESERCIZIARIO >**

54) *Trovare gli attori che hanno interpretato più personaggi in uno stesso film (+ di 1 !!)*

```
select distinct P1.Attore
from INTERPRETA P1 , INTERPRETA P2
where P1.Attore = P2.Attore
 and P1.Film = P2.Film
 and P1.Personaggio <> P2.Personaggio
```

*Soluzione  
"algebraica"*

**< ESERCIZIARIO >**

54) *Trovare gli attori che hanno interpretato più personaggi in uno stesso film (+ di 1 !!)*

```
select distinct Attore
from INTERPRETA
group by Attore, Film
having count(*) > 1
```

**PIÙ EFFICIENTE**

*Tipicamente riesce a sfruttare  
un **indice** definito sulla chiave  
per raggruppare rapidamente*

*select Attore as Chi, Film as Dove, count(\*) as Quanti*

116

**< ESERCIZIARIO >**

*55) Trovare i film in cui recita un solo attore e vi interpreta più personaggi*

select **P1.Film**

from INTERPRETA P1, INTERPRETA P2

where P1.Film = P2.Film and P1.Attore = P2.Attore  
and P1.Personaggio <> P2.Personaggio

**except**

select M.Film

from INTERPRETA M, INTERPRETA P

where M.Film=P.Film and M.Attore<>P.Attore

117

**< ESERCIZIARIO >**

*La soluzione precedente corrisponde, in algebra, a*

$$\Pi_{\text{Film}} ( \text{INT} \bowtie_{\text{Attore}=\text{Attore} \wedge \text{Film}=\text{Film} \wedge \text{Person} \neq \text{Person}} \text{INT} ) - \Pi_{\text{Film}} ( \text{INT} \bowtie_{\text{Film}=\text{Film} \wedge \text{Attore} \neq \text{Attore}} \text{INT} )$$

*SQL permette anche di ragionare sui gruppi:*

```
select Film
from INTERPRETA
group by Film
having count(*) > 1 and count(distinct Attore) = 1
```

118

**< ESERCIZIARIO >**

56) Attori italiani che non hanno mai recitato con altri italiani

select Nome

from ATTORE **A1**

where Nazionalità = 'Italiana' and not exists (

select \*

from INTERPRETA I1, INTERPRETA I2, ATTORE A2

where I1.Titolo = I2.Titolo and

I2.Attore = A2.Nome and I1.Attore = **A1.Nome** and

A2.Nazionalità = 'Italiana' and

**A1.Nome <> A2.Nome** )

*Deve essere un'altra persona...*<sup>119</sup>

< **ESERCIZIARIO** >



## 56) Attori italiani che non hanno mai recitato con altri italiani

*In alternativa si possono definire opportune viste intermedie*

```
select Attore
from Interp-italiano
```

**except**

```
select X.Attore
from Interp-italiano X,
 Interp-italiano Y
where X.Film = Y.Film and
 X.Nome <> Y.Nome
```

```
create VIEW Attore-italiano as
select Nome from ATTORE
where Nazionalità="Italiana"
```

```
create view Interp-italiano as
select Film, Attore
from INTERPRETA
where Attore in (select Nome
 from Attore-italiano)
```

## 57) *Film di registi italiani in cui non recita nessun italiano*

```
select Titolo
from FILM join REGISTA on Nome = NomeRegista
where Nazionalità = Italiana
except
select NomeFilm
from INTERPRETA join ATTORE on Nome=NomeAttore
where Nazionalità = Italiana
```

## 57) *Film di registi italiani in cui non recita nessun italiano*

```
select Titolo
from FILM join REGISTA on Nome=NomeRegista
where Nazionalità = Italiana and
 Titolo not in (select NomeFilm
 from INTERPRETA join ATTORE
 on Nome = NomeAttore
 where Nazionalità = Italiana)
```

*58) Registi che hanno recitato in (almeno) uno dei film che hanno diretto*

```
select distinct NomeRegista
from FILM join INTERPRETA
 on Titolo=Film
where NomeRegista = Attore
```

**< ESERCIZIARIO >**

60) *I registi che hanno recitato in **tutti** i loro film*

```
select NomeRegista
from REGISTA R
where not exists (
 select *
 from FILM F
 where not exists (
 select *
 from INTERPRETA
 where R.Nome = F.Regista and
 Film = F.Titolo and R.Nome = Attore
```

124

< **ESERCIZIARIO** >

**4)** *Il seguente schema relazionale descrive i campionati di pallavolo (non sono possibili i pareggi). La classifica è “fotografata” per ogni squadra alla fine di ogni giornata di ogni campionato (Anno e Giornata, infatti, appartengono alla chiave)*

Partita( Anno, Giornata, SquadraCasa,  
SquadraOspite, SetVintiCasa, SetVintiOspite)

Classifica( Anno, Giornata, Squadra, TotPunti, Posizione )

Squadra ( Nome, Città, NomeStadio )

*35) Formulare una query SQL che restituisca la squadra con il massimo numero di vittorie*

```
create view Vittoria (Anno, Giorn, Squadra) as
(select Anno, Giornata, SquadraCasa
 from Partita
 where SetVintiCasa > SetVintiOspite
 union
 select Anno, Giornata, SquadraOspite
 from Partita
 where SetVintiOspite > SetVintiCasa)
```

**< ESERCIZIARIO >**

35) *Formulare una query SQL che restituisca la squadra con il massimo numero di vittorie*

```
create view Vittoria (Anno, Giorn, Squadra) as
(select Anno, Giornata, SquadraCasa
 from Partita
 where SetVintiCasa = 3
 union (all?)
 select Anno, Giornata, SquadraOspite
 from Partita
 where SetVintiOspite = 3)
```

*all **NON** è necessario (a sx dell'operatore c'è la chiave di Partita, e per la parte destra si confida nell'integrità dei dati: ogni squadra "gioca una sola volta" in ogni giornata)*

127

< **ESERCIZIARIO** >



*35) Formulare una query SQL che restituisca la squadra con il massimo numero di vittorie*

*Usando la view precedente:*

```
select Squadra
from Vittoria
group by Squadra
having count(*) >= all (select count(*)
 from Vittoria
 group by Squadra)
```

128

**< ESERCIZIARIO >**

36) Definire in SQL un comando di aggiornamento che assegni all'attributo TotPunti il saldo dei punti fino alla giornata corrente

```
update Classifica C
set TotPunti=(select Count(*) * 2
 from Vittorie V
 where V.Anno = C.Anno
 and V.Giornata <= C.Giornata
 and V.Squadra = C.Squadra)
```

129

< **ESERCIZIARIO** >

*37) Definire un comando che assegni all'attributo Posizione la posizione in classifica della squadra, assumendo che TotPunti già rappresenti i punti conquistati dalla squadra fino a quella giornata (inclusa), tenendo conto SOLO dei "punti vittoria"*

La posizione è (in base a *questa* definizione) pari a 1 più il numero di squadre che a quella giornata hanno un maggiore numero totale di punti. Così si tratta correttamente anche il parimerito

37) Definire un comando che assegni all'attributo *Posizione* la posizione in classifica della squadra, assumendo che *TotPunti* già rappresenti i punti conquistati dalla squadra fino a quella giornata (inclusa), tenendo conto SOLO dei "punti vittoria"

update Classifica **C**

set Posizione = **1** + ( select count(\*)

from Classifica C1

where C1.Anno = **C**.Anno

and C1.Giornata = **C**.Giornata

and C1.TotPunti > **C**.TotPunti )

131

< **ESERCIZIARIO** >

**7)** Partite dei campionati di serie A, B, C. Ad ogni *partita giocata* è assegnato un codice univoco (per comodità).  
Ogni partita è (ovviamente) giocata da 2 squadre

**Partita**(Codice, Campionato, Serie, Data, Stadio)

Risultato ( Squadra, Partita, Punti, Reti )

*Si noti che la registrazione dei punti è **ridondante**, in quanto deducibile confrontando le reti delle squadre di una partita, ma molto comoda (occorrono 2 join per confrontare le reti)* 132

< **ESERCIZIARIO** >

*45) Le squadre che hanno giocato sempre e solo in serie A*

```
select distinct Squadra
from Risultato
where Squadra not in (select Squadra
 from Risultato, Partita
 where Partita = Codice
 and Serie <> "A")
```

**< ESERCIZIARIO >**

46) *La squadra che ha conquistato più punti in un campionato di serie A*

*Vista intermedia: di ogni squadra calcola i punti in ogni campionato*

create view PuntiTotInA(Camp, Squadra, PTot) AS

( select Campionato, Squadra, **sum(Punti)**

from Partita, Risultato

where Codice=Partita and Serie="A"

group by Squadra, Campionato )

select Squadra, PTot

from PuntiTotInA

where PTot =

( select **max(PTot)**

from PuntiTotInA )

< **ESERCIZIARIO** >

47) Il calendario, *con i risultati*, delle 34 giornate del campionato di serie A '01/'02 in ordine cronologico

Attenzione:  $34 * 9 = 306$  tuple (1  $\forall$  partita)

N.B. Lo schema non permette di stabilire con certezza (né di calcolare per tutte le squadre) chi ha giocato in casa; includiamo allora nel risultato anche lo **Stadio**, che può aiutare a capirlo.

Però resta ancora il problema dei derby e dei campi neutri... quindi imponiamo di visualizzare *prima* il nome della *vincitrice*, e in caso di *pareggio* di adottare, ad esempio, il criterio *lessicografico*

< **ESERCIZIARIO** >



47) Il calendario, *con i risultati*, delle 34 giornate del campionato di serie A '01/'02 in ordine cronologico

```
select Data, r1.Squadra, r2.Squadra,
 r1.Reti, r2.Reti, Stadio
from Partita, Risultato r1, Risultato r2
where Serie = 'A' and Campionato = '01-02'
 and r1.Partita = Codice and r2.Partita = Codice
 and (r1.Reti > r2.Reti OR
 (r1.Reti = r2.Reti and r1.Squadra < r2.Squadra))
order by Data
```

136

< **ESERCIZIARIO** >

*Con uno schema diverso è MOLTO più facile*

Partita ( SqCasa, SqOsp, Data, RetiCasa, RetiOsp,  
Campionato, Serie)

Il calendario, *con i risultati*, delle 34 giornate del  
campionato di serie A '01/'02 in ordine cronologico

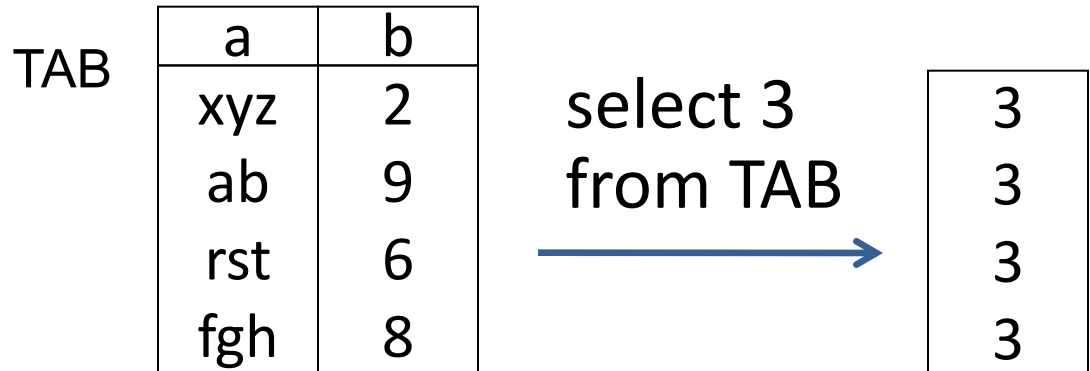
```
select Data, SqCasa, SqOsp, RetiCasa
from Partita
where Serie = 'A' and Campionato = '01-02'
order by Data
```

49) Sul **secondo** schema: calcolare la classifica del campionato di A del '01-'02 in modo che lo schema del risultato R sia:

R ( Squadra, PuntiTot ) ...in ordine di punteggio

Usiamo una vista che costruisca una tabella per assegnare a ogni squadra per ogni partita il punteggio corretto ( 3, 1, 0 )

N.B.  
Sono ammesse  
*costanti*  
nelle target list:



```
create view PuntiFatti (Squadra, Punti) as
(select SqCasa, 3 from Partita
 where RetiCasa > RetiOsp and Serie='A' and Camp='01/02'
 UNION ALL
 select SqOsp, 3 from Partita
 where RetiCasa < RetiOsp and Serie='A' and Camp='01/02'
 UNION ALL
 select SqCasa, 1 from Partita
 where RetiCasa = RetiOsp and Serie='A' and Camp='01/02'
 UNION ALL
 ...
```

...

UNION ALL

select SqOsp, **1** from Partita  
where RetiCasa = RetiOsp and Serie='A' and Camp='01/02'

UNION ALL

select SqCasa, **0** from Partita  
where RetiCasa < RetiOsp and Serie='A' and Camp='01/02'

UNION ALL

select SqOsp, **0** from Partita  
where RetiCasa > RetiOsp and Serie='A' and Camp='01/02' )

```
select Squadra, sum(Punti) as PuntiTot
from PuntiFatti
group by Squadra
order by 2 desc
```

*Nella vista è **necessario** assegnare separatamente i punteggi alle squadre di casa e alle squadre ospiti.*

*L'**ALL** è **necessario** perché gli operatori insiemistici eliminano i duplicati – diversamente “sopravviverebbero” al massimo un pareggio e una vittoria per ogni squadra.*

*Nella vista, anche assegnare 0 punti per le sconfitte è **necessario**, per includere in classifica una eventuale squadra che abbia sempre perso.*

*Si noti infine che l'ordinamento nella query è condotto su un valore aggregato (identificato posizionalmente, non essendo un attributo con un nome), e che la target list è **compatibile** con la clausola group by*

# TDE – 09/09/2013

CARTA (IdCARTA, COGNOME, NOME, RESIDENZA)

DISTRIBUTORE (IdDISTRIB, TITOLARE, CITTA)

RIFORNIMENTO (IdCARTA, DATA, ORA, IdDISTRIB, COMBUSTIBILE, LITRI, IMPORTO)

- Estrarre Nome e Cognome dei clienti che hanno effettuato il più alto numero di rifornimenti
- Trovare l'IdCarta dei clienti che hanno (complessivamente) acquistato una quantità di gasolio pari a quella della benzina (con una tolleranza di 20 litri)
- Stilare la classifica dei clienti (Cognome, Nome) a partire dal cliente con consumo maggiore in litri di combustibile
- Estrarre i clienti (Cognome, Nome) che hanno fatto rifornimenti di benzina (Combustibile="benzina") **solo** nella loro città di *residenza* e rifornimenti di gasolio **solo** in città *diverse* dalla loro città di residenza



- Estrarre Nome e Cognome dei clienti che hanno effettuato il più alto numero di rifornimenti

CARTA (IdCarta, COGNOME, NOME, RESIDENZA)  
DISTRIBUTORE (IdDistrib, TITOLARE, CITTA)  
RIFORNIMENTO (IdCarta, DATA, ORA, IdDistrib,  
COMBUSTIBILE, LITRI, IMPORTO)

```
select IdCarta, Nome, Cognome
from Rifornimento R join Carta C on R.IdCarta=C.IdCarta
group by IdCarta, Nome, Cognome
having count(*) >= ALL (select count(*)
 from Rifornimento
 group by IdCarta)
```

- Trovare l'IdCarta dei clienti che hanno (complessivamente) acquistato una quantità di gasolio pari a quella della benzina (con tolleranza di 20 litri)

CARTA (IdCarta, COGNOME, NOME, RESIDENZA)  
DISTRIBUTORE (IdDistrib, TITOLARE, CITTA)  
RIFORNIMENTO (IdCarta, DATA, ORA, IdDistrib,  
COMBUSTIBILE, LITRI, IMPORTO)

```
create view tG (idg, lg) as
select IdCarta, sum(Litri)
from Rifornimento
where Combustibile = "gasolio"
group by IdCarta
```

```
create view tB (idb, lb) as
select IdCarta, sum(Litri)
from Rifornimento
where Combustibile = "benzina"
group by IdCarta
```

```
select IdCarta
from tG join tB on idg = idb
where lg > lb-20 and lb > lg-20
```

*oppure:*

*lg-lb < 20 and lb-lg > -20*

*lg-lb between 0 and 20 or  
lb-lg between 0 and 20*

*lg-lb between -20 and 20*

- Stilare la classifica dei clienti (Cognome, Nome) a partire da quello col maggior consumo in litri

CARTA (IdCARTA, COGNOME, NOME, RESIDENZA)  
DISTRIBUTORE (IdDISTRIB, TITOLARE, CITTA)  
RIFORNIMENTO (IdCARTA, DATA, ORA, IdDISTRIB,  
COMBUSTIBILE, LITRI, IMPORTO)

*Riusando le viste tG e tB*

```
select Nome, Cognome, IdCarta, lg+lb as ConsumoTotale
from tG join tB on idg = idb join Carta on idb = IdCarta
order by 4
```

- Estrarre i clienti (Cognome, Nome) che hanno fatto rifornimenti di benzina (Combustibile="benzina") **solo** nella loro città di *residenza* e rifornimenti di gasolio **solo** in città *diverse* dalla loro città di residenza

CARTA (IdCarta, COGNOME, NOME, RESIDENZA)  
DISTRIBUTORE (IdDistrib, TITOLARE, CITTA)  
RIFORNIMENTO (IdCarta, DATA, ORA, IdDistrib,  
COMBUSTIBILE, LITRI, IMPORTO)

```
select Nome, Cognome
from (Carta C natural join Rifornim R1) join Rifornim R2 using(IdCarta)
where R1.Combustibile = "gasolio" and R2.Combustibile = "benzina"
and 0 = (select count(*)
 from Rifornimento natural join Distributore
 where Combustibile = "benzina" and
 IdCarta = C.IdCarta and Citta <> C.Residenza)
and not exists(select *
 from Rifornimento natural join Distributore
 where Combustibile = "gasolio" and
 IdCarta = C.IdCarta and Citta = C.Residenza)
```

- Estrarre i clienti (Cognome, Nome) che hanno fatto rifornimenti di benzina (Combustibile="benzina") **solo** nella loro città di *residenza* e rifornimenti di gasolio **solo** in città *diverse* dalla loro città di residenza

CARTA (IdCarta, COGNOME, NOME, RESIDENZA)  
DISTRIBUTORE (IdDistrib, TITOLARE, CITTA)  
RIFORNIMENTO (IdCarta, DATA, ORA, IdDistrib,  
COMBUSTIBILE, LITRI, IMPORTO)

```
select Nome, Cognome
from (Carta C natural join Riforn R1) join Riforn R2 using(IdCarta)
where R1.Combustibile = "gasolio" and R2.Combustibile = "benzina"
 and IdCarta not in (select IdCarta
 from Rifornimento natural join Distributore
 where Combustibile = "benzina" and
 Citta <> C.Residenza)
 and IdCarta not in (select IdCarta
 from Rifornimento natural join Distributore
 where Combustibile = "gasolio" and
 Citta = C.Residenza)
```

- Estrarre i clienti (Cognome, Nome) che hanno fatto rifornimenti di benzina (Combustibile="benzina") **solo** nella loro città di *residenza* e rifornimenti di gasolio **solo** in città *diverse* dalla loro città di residenza

CARTA (IDCARTA, COGNOME, NOME, RESIDENZA)  
DISTRIBUTORE (IDDISTRIB, TITOLARE, CITTA)  
RIFORNIMENTO (IDCARTA, DATA, ORA, IDDISTRIB,  
COMBUSTIBILE, LITRI, IMPORTO)

[illegible]

**L'orario dei treni e degli autobus** è rappresentato in assenza di fermate intermedie. Per semplicità, assumiamo che vi sia una sola stazione in ogni città e che la periodicità dell'orario sia giornaliera.

TRATTATRENO (IDTRENO, ORAPART, STAZPART, ORAARR, STAZARR)

STAZIONE (CITTA, REGIONE)

**TDE 23/9/2013**

TRATTABUS (IDBUS, ORAPART, STAZPART, ORAARR, STAZARR)

### **SQL (12 punti)**

1. Estrarre la città da cui parte il maggior numero di tratte (contando quelle di treno sia quelle di autobus). (3 p.)
2. Estrarre tutti i collegamenti tra città lombarde raggiungibili in treno con al più un cambio di treno. (3 p.)
3. Estrarre le coppie di città della stessa regione, raggiungibili con un tratto di treno seguito da uno di autobus che parta entro quindici minuti dall'arrivo del treno, ma che non siano collegate direttamente via treno. (3 p.)
4. Estrarre il tempo minimo necessario per raggiungere Mantova da Milano partendo alle 14:00 e usando solo treni oppure solo autobus, e comunque con NON PIU' di UN cambio. (3 p.)

### **Linguaggi Formali (6 punti)**

5. Esprimere la seconda interrogazione in Algebra Relazionale oppure in Calcolo Relazionale. (3 p.)
6. Estrarre in Datalog l'insieme delle città raggiungibili in treno da Milano con un numero arbitrario di cambi. Si può esprimere la stessa interrogazione in Algebra? (3 p.)

TRATTATRENO (IDTRENO, ORAPART, STAZPART, ORAARR, STAZARR)

STAZIONE (CITTA, REGIONE)

TRATTABUS (IDBUS, ORAPART, STAZPART, ORAARR, STAZARR)

1. Estrarre la città da cui parte il maggior numero di tratte (contando quelle di treno sia quelle di autobus). (3 p.)

create view Tratta ( ID, OraPart, StazPart, OraArr, StazArr ) as

( select \*

from TrattaTreno

union

select \*

from TrattaBus )

select StazPart, count(\*)

from Tratta

group by StazPart

having count(\*) >= ALL ( select count(\*)

from Tratta

group by StazPart )



TRATTATRENO (IDTRENO, ORAPART, STAZPART, ORAARR, STAZARR)  
STAZIONE (CITTA, REGIONE)  
TRATTABUS (IDBUS, ORAPART, STAZPART, ORAARR, STAZARR)

2. Estrarre tutti i collegamenti tra città lombarde raggiungibili in treno con al più un cambio di treno. (3 p.)

```
select T.StazPart, T.StazArrivo
from (TrattaTreno T join Stazione S1 on S1.Citta = T.StazPart)
 join Stazione S2 on S2.Citta = T.StazArr
where S1.Regione = "Lombardia" and S1.Regione = "Lombardia"
union
select A.StazPart, B.StazArrivo
from ((TrattaTreno A join TrattaTreno B on A.StazArrivo = B.StazPart)
 join Stazione S1 on S1.Citta = A.StazPart)
 join Stazione S2 on S2.Citta = B.StazArr
where S1.Regione = "Lombardia" and S1.Regione = "Lombardia"
and S1.Citta <> S2.Citta and A.OraArr < B.OraPart
```

TRATTATRENO (IDTRENO, ORAPART, STAZPART, ORAARR, STAZARR)

STAZIONE (CITTA, REGIONE)

TRATTABUS (IDBUS, ORAPART, STAZPART, ORAARR, STAZARR)

3. Estrarre le coppie di città della stessa regione, raggiungibili con un tratto di treno seguito da uno di autobus che parta entro quindici minuti dall'arrivo del treno, ma che non siano collegate direttamente via treno. (3 p.)

```
select S1.Citta, S2.Citta
from Stazione S1 join TrattaTreno T on S1.Citta=T.StazPart
 join TrattaBus B on T.StazArrivo=B.StazPart
 join Stazione S2 on B.StazArrivo=S2.Citta
where S1.Regione = S2.Regione and S1.Citta <> S2.Citta and
 B.OraPart – T.OraArr between 1min and 15min and
not exists(select * from TrattaTreno X
 where X.StazPart = S1.Citta and X.StazArr = S2.Citta)
```

TRATTATRENO (IDTRENO, ORAPART, STAZPART, ORAARR, STAZARR)  
STAZIONE (CITTA, REGIONE)  
TRATTABUS (IDBUS, ORAPART, STAZPART, ORAARR, STAZARR)

4. Estrarre il tempo minimo necessario per raggiungere Mantova da Milano partendo alle 14:00 e usando solo treni oppure solo autobus, e comunque con NON PIU' di UN cambio. (3 p.)

```
select min(X.OraArr – 14:00)
from Tratta X
where X.StazArr = "Mantova" and
(X.StazPart = Milano and X.OraPart = 14:00
OR
exists (select * from Tratta TP
 where TP.StazPart = Milano and TP.OraPart = 14:00 and TP.Tipo = X.Tipo and
 TP.OraArr < X.OraPart and TP.StazArr X.StazPart))

create view Tratta (ID, OraPart, StazPart, OraArr, StazArr, Tipo) as
(select *, "T"
 from TrattaTreno
 union
 select *, "B"
 from TrattaBus)
```

TRATTATRENO (IDTRENO, ORAPART, STAZPART, ORAARR, STAZARR)

STAZIONE (CITTA, REGIONE)

TRATTABUS (IDBUS, ORAPART, STAZPART, ORAARR, STAZARR)

5. Esprimere la seconda interrogazione in Algebra Relazionale oppure in Calcolo Relazionale. (3 p.)

Algebra Relazionale : si usa la stessa struttura dell'SQL (join e union dei due casi)

$$\begin{aligned} & \{ t \mid t_{SPa} \in \text{STAZIONE}, t_{SAr} \in \text{STAZIONE} \\ & \quad t_{SPa}[\text{Regione}] = \text{"Lombardia"} \wedge t_{SAr}[\text{Regione}] = \text{"Lombardia"} \wedge t_{SAr}[\text{Regione}] = \text{"Lombardia"} \wedge \text{DIV+ORA} \\ & \quad ( ( \exists t_{TDir} \in \text{TrattaTreno} \mid \\ & \quad \quad t_{TDir}[\text{StazPart}] = t_{SPa}[\text{Citta}] \wedge t_{TDir}[\text{StazArr}] = t_{SAr}[\text{Citta}] ) \\ & \quad \vee \\ & \quad ( \exists t_{T1} \in \text{TrattaTreno}, \exists t_{T2} \in \text{TrattaTreno} \mid \\ & \quad \quad t_{T1}[\text{StazPart}] = t_{SPa}[\text{Citta}] \wedge t_{T1}[\text{StazArr}] = t_{T2}[\text{StazPart}] \wedge t_{T1}[\text{OraArr}] < t_{T2}[\text{OraPart}] \wedge t_{T2}[\text{StazArr}] = t_{SAr}[\text{Citta}] ) ) \\ & \} \end{aligned}$$

TRATTATRENO (IDTRENO, ORAPART, STAZPART, ORAARR, STAZARR)

STAZIONE (CITTA, REGIONE)

TRATTABUS (IDBUS, ORAPART, STAZPART, ORAARR, STAZARR)

6. Estrarre in Datalog l'insieme delle città raggiungibili in treno da Milano con un numero arbitrario di cambi. Si può esprimere la stessa interrogazione in Algebra? (3 p.)

Tratta( C1, C2 ) :- TrattaTreno( \_, \_, C1, \_, C2 )

Tratta( C1, C2 ) :- TrattaBus( \_, \_, C1, \_, C2 )

**RaggDaMi**( Citta ) :- Tratta("Milano", Citta )

**RaggDaMi**( Citta ) :- **RaggDaMi**( X ),

Tratta( X, Citta ),

Citta != "Milano"

# Concerti (TDE '06)

*Il seguente schema rappresenta un insieme di orchestre in relazione ai loro concerti e alle sale in cui tali concerti vengono tenuti:*

ORCHESTRA ( CodOrchestra, NomeO, NomeDirettore,  
NumElementi, AnnoFondata )

CONCERTO ( CodC, Data, CodO, CodS, PrezzoBiglietto )

SALA ( CodSala, NomeS, Città, Capienza )

*Trovare il codice e il nome delle orchestre con più di 30 elementi che hanno tenuto concerti sia a Genova, sia a Milano e non hanno mai tenuto concerti a Bologna*

```
select CodOrchestra, NomeO
```

```
from ORCHESTRA
```

```
where NumElementi > 30 and
```

```
 CodOrchestra in (select CodO from Concerto join Sala on CodS=CodSala
 where Città='Milano') and
```

```
 CodOrchestra in (select CodO from Concerto join Sala on CodS=CodSala
 where Città='Genova') and
```

```
 CodOrchestra not in (select CodO from Concerto join Sala on CodS=CodSala
 where Città='Bologna')
```

*Scrivere un comando SQL che aumenta del 10% il prezzo del biglietto dei concerti di quelle orchestre che sono state fondate prima del 2000, che hanno tenuto in totale nella loro vita più di 100 concerti e il cui prezzo medio dei biglietti sia inferiore ai 20 euro*

update Concerti

set PrezzoBiglietto = PrezzoBiglietto \* 1.1

where CodO in

( select CodOrchestra

from Orchestra join Concerti on CodOrchestra = CodO

where AnnoFondata < 2000

group by CodOrchestra

having count(\*) > 100 and avg(prezzobiglietto) < 20 )



*I nomi delle orchestre che hanno tenuto un concerto in tutte le sale.  
(Riformulabile come: data un'orchestra O, non esista una sala in cui O  
non ha tenuto almeno un concerto)*

```
select NomeO
from Orchestra O
where not exists (select *
 from SALA S
 where not exists (select *
 from CONCERTO C
 where C.CodS = S.CodSala and
 C.CodO = O.CodOrchestra))
```