

Esercizio 1:

Sia dato il seguente schema relazionale relativo a maratone internazionali:

Maratoneta(CodiceMaratoneta, Nome, Cognome, MigliorePrestazione);

Maratona(Nome, Città);

SiTrovaIn(Città, Nazione);

EdizioniMaratona(Nome, Anno);

Vincitore(Athlete, Maratona, Anno).

Si assume che ogni maratoneta sia identificato univocamente da un codice e sia caratterizzato dal nome, dal cognome e dalla miglior prestazione da lui ottenuta in una maratona (si assume, per semplicità, che il tempo impiegato da un maratoneta sia espresso in secondi). Si assume che ogni maratona sia identificata dal suo nome e sia caratterizzata dalla città in cui ha luogo (non si escluda la possibilità che in una stessa città possono esservi due o più maratone diverse). Ad ogni città sia associata la nazione cui appartiene (si assume, per semplicità, che non esistano città con lo stesso nome). Si assume che ogni edizione di una certa maratona sia identificata dal nome e dall'anno in cui si svolta. Infine, si assume che la relazione *Vincitore* registri i vincitori delle diverse edizioni delle maratone considerate.

Definire preliminarmente le chiavi primarie, le eventuali altre chiavi candidate e, se ve ne sono, le chiavi esterne delle relazioni date. Successivamente, formulare opportune interrogazioni in SQL che permettano di determinare (senza usare l'operatore CONTAINS e usando solo se necessario le funzioni aggregate):

(a) i maratoneti che hanno vinto almeno una maratona tedesca, ma nessuna maratona italiana;

(b) i maratoneti che hanno vinto tutte le edizioni di (almeno) una maratona.

(FACOLTATIVO) Formulare un'interrogazione in algebra relazionale per il punto (b), senza usare l'operatore di divisione e usando solo se necessario le funzioni aggregate.

a) $\text{SELECT Distinct Athlete}$

$\text{FROM Vincitore V1, Maratona M1, SiTrovaIn SI}$

$\text{WHERE V1.Nazione = M1.Nazione AND M1.Città = SI.Città AND SI.Nazione = 'Germania'}$
 $\text{AND V1.Athlete NOT IN (SELECT Distinct Athlete}$

$\text{FROM Vincitore V2, Maratona M2, SiTrovaIn S2}$

$\text{WHERE V2.Nazione = M2.Nazione AND M2.Città = S2.Città AND S2.Nazione = 'Italia')}$

b) SELECT M1.*

$\text{FROM Maratona M1, Maratona M2}$

$\text{WHERE NOT EXISTS (SELECT *}$

FROM Vincitore V1

$\text{WHERE M2.Nome = V1.Maratona AND V1.Athlete <> M1.CodiceAthlete)}$

b2) $\text{Maratone-No-food} \leftarrow$

$\Pi_{\text{Maratone}} \left(\text{Vincitore, } \begin{array}{l} \text{X Vincitore} \\ \text{nazione} = \text{Maratona} \\ \wedge \text{vincitore} < \text{Vincitore} \end{array} \right)$

$\text{Maratone-food} \leftarrow \Pi_{\text{Maratone}} (\text{Maratone}) - \text{Maratone-No-food}$

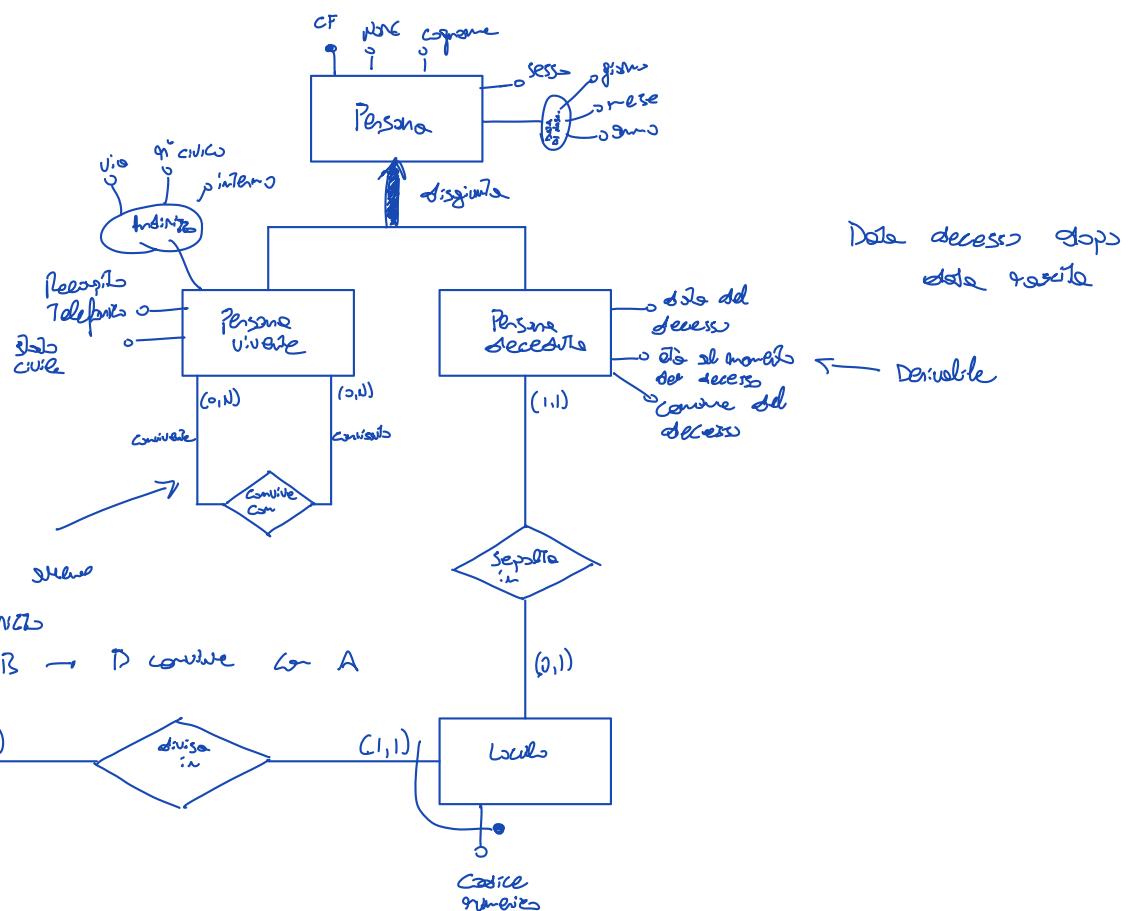
$R \leftarrow \Pi_{\text{Athlete}} (\text{Vincitore } \begin{array}{l} \text{X Maratone-food} \\ \text{nazione} = \text{Maratona} \end{array})$

Esercizio 2:

Sia dato il seguente insieme di requisiti relativi ad una base di dati che regista informazioni sulle persone che risiedono o sono sepolte in un dato comune italiano.

- Di ogni persona interessano nome, cognome, codice fiscale, sesso (m,f) e data di nascita (giorno, mese, anno). Una persona può essere residente nel comune (vivente) o sepolta nel comune (deceduta).
- Di ogni persona vivente interessano indirizzo (via, numero civico, interno), recapito telefonico, stato civile (celibe (nubile), coniugato(a), vedovo(a), separato(a), divorziato(a), **deceduto(a)**) e familiari conviventi.
- Si assume, per semplicità, che il comune abbia un solo cimitero suddiviso in diverse aree, ciascuna identificata da un codice alfanumerico. All'interno di ciascuna area, ogni loculo sia identificato univocamente da un codice numerico (non si escluda la possibilità che loculi situati in aree diverse possano avere lo stesso numero).
- Di ogni persona sepolta nel cimitero del comune interessano loculo, data del decesso, età al momento del decesso e comune del decesso.

Si definisca uno schema Entità-Relazioni che descriva il contenuto informativo del sistema, illustrando con chiarezza le eventuali assunzioni fatte. Lo schema dovrà essere completato con attributi ragionevoli per ciascuna entità (identificando le possibili chiavi) e relazione. Vanno specificati accuratamente i vincoli di cardinalità e partecipazione di ciascuna relazione. Si definiscano anche eventuali regole di gestione (regole di derivazione e vincoli di integrità) necessarie per codificare alcuni dei requisiti attesi del sistema.



Esercizio 3:

Si immagini di voler progettare una base di dati che memorizza informazioni relative alle squadre partecipanti ad un campionato di pallavolo. La base di dati è costituita da due tabelle: **squadre** e **giocatori**.

La tabella **squadre** è caratterizzata dai seguenti attributi: **nome** (stringa di al più 50 caratteri; è la chiave primaria), **anno_fondazione** (specifica quando la squadra è stata creata; può essere nullo) e **capitano** (identificativo del giocatore che ha il ruolo di capitano nella stagione corrente; è chiave esterna verso **giocatori**, unica e non nulla). La tabella **giocatori** è caratterizzata dai seguenti attributi: **num_cartellino** (stringa di esattamente 5 caratteri; è la chiave primaria), **nome** (nome del giocatore, non nullo), **ingaggio** (ingaggio annuale del giocatore, non nullo) e **squadra** (la squadra a cui il giocatore appartiene; è una chiave esterna che fa riferimento alla chiave primaria di **squadre** e può essere nulla nel caso di giocatori temporaneamente svincolati).

Si scriva del codice SQL per creare (ma non popolare) le seguenti tabelle, usando dei tipi di attributo ragionevoli in tutti quei casi in cui non siano stati esplicitamente specificati:

Table 1: **squadre**

nome	anno_fondazione	capitano
SiamoFortissimi	-	c1111
ANoiChiCiBatte	1994	c2222
PerdiamoSempre	2020	c3333

Table 2: **giocatori**

num_cartellino	nome	ingaggio	squadra
c1111	Tizio	105000	SiamoFortissimi
c2222	Caio	100000	ANoiChiCiBatte
c3333	Sempronio	80000	PerdiamoSempre
c4444	Mevio	85000	PerdiamoSempre
c5555	Filano	60000	-
c6666	Calpurnio	75000	SiamoFortissimi

Si ipotizzi di voler inserire un nuovo giocatore (*PincoPallo*) e la relativa squadra (*IDisperati*) (non presente nella base di dati). Si specifichi, se esiste, il corretto ordine delle operazioni necessarie per effettuare i due inserimenti, motivando la risposta (non è necessario scrivere il codice SQL).

Inoltre, si assuma che durante la sessione di mercato il giocatore *Calpurnio* sia stato venduto dalla squadra *SiamoFortissimi* alla squadra *ANoiChiCiBatte*. Si scriva il codice SQL che effettua tale modifica.

Infine, si immagini che al termine della stagione, la squadra *PerdiamoSempre* sia acquisita dalla squadra *SiamoFortissimi*. Si scriva il codice SQL per spostare tutti i giocatori dalla squadra *PerdiamoSempre* alla squadra *SiamoFortissimi* ed eliminare la squadra *PerdiamoSempre* dalla base di dati.

Si consideri il seguente vincolo: una squadra di pallavolo deve essere costituita da al più 13 giocatori e almeno 6. Quali operazioni, e su quali tabelle, possono violare questo vincolo? Si scelga una di queste operazioni e si scriva un trigger SQL che eviti tale violazione.

a) creare Tabella **squadre**

name varchar(50) primary key,
anno_fondazione date,
capitano char(5) unique not null

)

creare Tabella **giocatori**

num_cartellino char(5) primary key,
name varchar(20) not null,
ingaggio integer not null check (ingaggio > 0),
squadra varchar(50) references squadre

)

Altre Tabelle **squadre** sono costituite fk-squadre foreign key capitano references giocatori

- b) d'ordine corretto è il seguente:
- si inserisce Pino Pello nella Tabella giocatori, con valore nullo per il campo squadra (che è un valore ammesso)
 - si inserisce la squadra 'Disperdi' nella Tabella squadre, con valore di chiave esterna 'capitano' uguale al valore 'non controllato' del giocatore Pino Pello
 - si modifica l'entry di Pino Pello nella Tabella giocatori, assegnandole alla squadra 'Disperdi'

c) UPDATE TABLE giocatori SET squadra = 'Anarchiaballe' WHERE nome = 'Colpozio'

d) Start Transaction

DELETE Tabla giocatori; SET squadra = 'SiamoFottissimi' WHERE squadra = 'PediamoSempre';
DELETE FROM Tabla squadre WHERE nome = 'PediamoSempre'

Commit

e) Le vincoli può essere violati da operazioni di inserimento, aggiornamento o cancellazione nella Tabella giocatori

Cancellazione:

```
CREATE TRIGGER controllo_nume_min()
BEFORE DELETE OR UPDATE ON giocatori
FOR EACH ROW
EXECUTE PROCEDURE numero_min();
```

CREATE OR REPLACE FUNCTION numero_min()

RETURNS TRIGGER LANGUAGE plpgsql AS \$\$

DECLARE n integer;

begin

```
SELECT count(*) INTO n
FROM giocatori g1
WHERE g1.squadra = old.squadra;
```

if n = 6 then

raise notice 'Una squadra non può avere più di 6 giocatori';

return null;

end if;

return old;

end;

\$\$

Esercizio 4:

Si stabilisca se i seguenti schedule appartengono o meno a VSR, CSR, 2PL e 2PL stretto:

1. $s_1 : r_3(y), r_3(z), r_1(x), w_1(x), w_3(y), w_3(z), r_2(z), r_1(y), w_1(y), r_2(y), w_2(y), r_2(x), w_2(x);$
2. (FACOLTATIVO) $s_2 : r_2(z), r_2(y), w_2(y), r_3(z), r_1(x), w_1(x), w_3(y), w_3(z), r_2(x), r_1(y), w_1(y), w_2(x).$

$s_1 \in USR?$:

$$\text{legge: } \{ (r_2(x), w_1(x)), (r_1(y), w_3(y)), (r_2(z), w_3(z)), (r_2(y), w_1(y)) \}$$

$$\text{sf: } \{ w_2(x), w_2(y), w_3(z) \}$$

- Da $(r_2(x), w_1(x)) \in \text{legge}$ lo che $t_1 < t_2$
- Da $(r_2(z), w_3(z)) \in \text{legge}$ lo che $t_3 < t_2$
- Da $(r_1(y), w_3(y)) \in \text{legge}$ lo che $t_3 < t_1$

Per questo schedule seriali che potrebbe essere equivalente è t_3, t_1, t_2 :

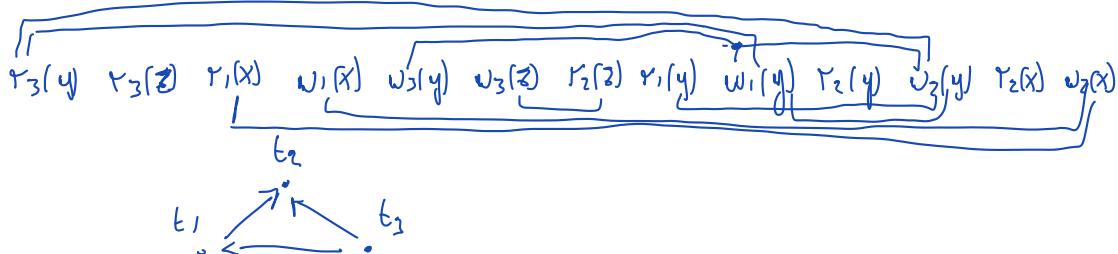
$$r_3(y) \ r_3(z) \ w_3(y) \ w_3(z) \ r_1(x) \cup \infty \ r_1(y) \ w_1(y) \ r_2(z) \ r_2(y) \ w_2(y) \ r_2(x) \ w_2(x)$$

$$\text{legge: } \{ (r_1(y), w_3(y)), (r_2(z), w_3(z)), (r_2(y), w_1(y)), (r_2(x), w_1(x)) \}$$

$$\text{sf: } \{ w_2(x), w_2(y), w_3(z) \}$$

Quindi: le due schedule sono equivalenti e $s_1 \in USR$

$s_1 \in CSR$:



Le opere è serializzabile quindi: $s_1 \in CSR$

• Si è ZPL:

