**Sistemi Informativi T**
**15 febbraio 2021**
**Risoluzione**

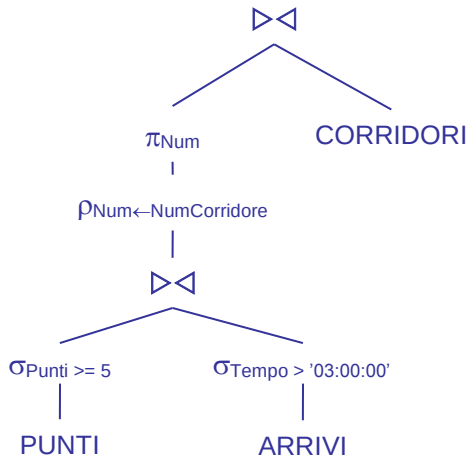
## 1) Algebra relazionale (3 punti totali):

Date le seguenti relazioni:

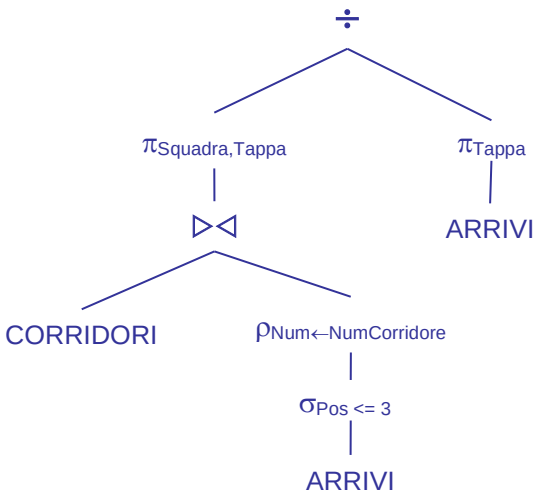
```
CORRIDORI(Num,Nome,Squadra);
PUNTI(Pos,Punti);
ARRIVI(Tappa,Pos,NumCorridore,Tempo),
NumCorridore REFERENCES CORRIDORI;
-- PUNTI.Pos è un intero che vale 1,2 o 3 e indica i punti per
-- le prime tre posizioni di ogni tappa, mentre ARRIVI.Pos può
-- assumere anche valori maggiori (per i quali non vi sono punti).
-- Tempo è di tipo TIME (formato 'hh:mm:ss').
```

si esprimano in algebra relazionale le seguenti interrogazioni:

**1.1) [1 p.]** I dati dei corridori che in almeno una tappa hanno ottenuto 5 punti o più con un tempo maggiore di 3 ore




**1.2) [2 p.]** Le squadre che in ogni tappa hanno avuto almeno un corridore nelle prime tre posizioni



**Sistemi Informativi T**
**15 febbraio 2021**
**Risoluzione**

## 2) SQL (5 punti totali)

Con riferimento al DB dell'esercizio 1, si esprimano in SQL le seguenti interrogazioni:

**2.1) [2 p.]** Per ogni corridore, il numero, il nome, i punti totali (0 se non ha punti) e la miglior posizione ottenuta, ordinando per valori decrescenti dei primi e, a parità, per valori crescenti della seconda

```
SELECT   C.Num, C.Nome, COALESCE(SUM(P.Punti),0) AS PuntiTotali,
                          MIN(A.Pos) AS MigliorPosizione
FROM     ARRIVI A LEFT JOIN PUNTI P ON (A.POS = P.POS)
               JOIN CORRIDORI C ON (A.NumCorridore = C.Num)
GROUP BY C.Num, C.Nome
ORDER BY PuntiTotali DESC, MigliorPosizione;

-- L'outer join serve per poter inserire nel risultato anche chi non
-- ha mai ottenuto punti. Per tali corridori SUM(P.Punti) è NULL,
-- e la funzione COALESCE restituisce il valore 0, come richiesto
```

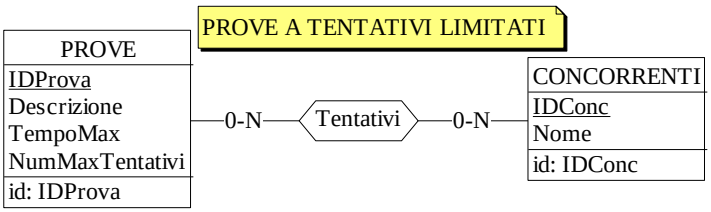
**2.2) [3 p.]** Considerando solo le prime 3 posizioni in ogni tappa, i dati del corridore che ha dato il maggior distacco di tempo al successivo

```
WITH DISTACCHI(NumCorridore,Distacco) AS (
   SELECT A1.NumCorridore, A2.Tempo - A1.Tempo
   FROM   ARRIVI A1, ARRIVI A2
   WHERE  A1.Pos + 1 = A2.Pos
   AND    A2.Pos <= 3
   AND    A1.Tappa = A2.Tappa)
SELECT   C.*
FROM     CORRIDORI C, DISTACCHI D
WHERE    C.Num = D.NumCorridore
AND      D.Distacco = ( SELECT   MAX(Distacco)
                        FROM     DISTACCHI );

-- La c.t.e. calcola i distacchi di tutte le tappe
```

**Sistemi Informativi T**
**15 febbraio 2021**
**Risoluzione**

## 3) Modifica di schema E/R e del DB (6 punti totali)

Dato il file ESE3.lun fornito, in cui è presente lo schema ESE3-input in figura:

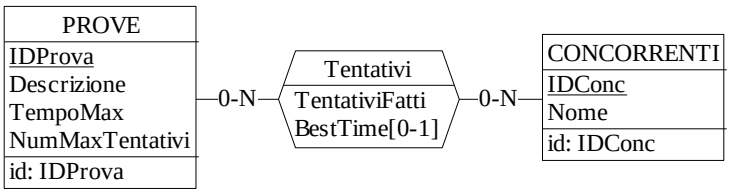



Specifiche aggiuntive:
Per ogni coppia (concorrente, prova) si tiene traccia del numero di tentativi fatti (TentativiFatti, default 0) e del miglior tempo ottenuto (BestTime, default NULL).
Quando BestTime <= TempoMax la prova si intende superata.

Trraduzione:
si traduca tutto.

Operazioni:
Si aggiorni il DB a seguito di un nuovo tentativo, ricavando dal DB il valore di TentativiFatti.
Si assume che l'inserimento con i valori di default per la relativa coppia (concorrente,prova) sia già stato fatto.
Si impediscano aggiornamenti se la prova è stata superata o se si è raggiunto il numero massimo di tentativi per quella prova.

**3.1) [2 p.]** Si produca uno schema ESE3-modificato secondo le Specifiche aggiuntive;




**3.2) [1 p.]** Si copi lo schema ESE3-modificato in uno schema ESE3-tradotto. Mediante il comando Transform/Quick SQL, si traduca la parte di schema specificata, modificando lo script SQL in modo da essere compatibile con DB2 e permettere l'esecuzione del punto successivo, ed eventualmente aggiungendo quanto richiesto dalle Specifiche aggiuntive;
Si veda il relativo file .sql

**3.3) [3 p.]** Si scriva l'istruzione SQL che modifica il DB come da specifiche (usare valori a scelta) e si definiscano i trigger necessari.

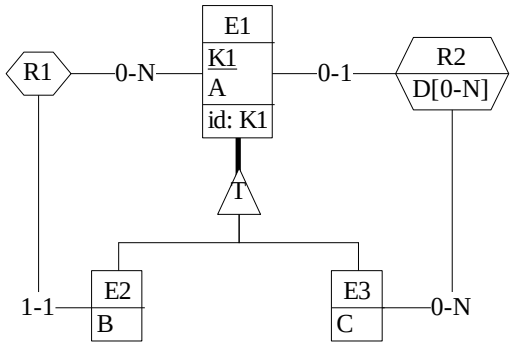
```
UPDATE Tentativi
SET BestTime = MIN(:nuovotempo,COALESCE(BestTime,:nuovotempo)),
      TentativiFatti = TentativiFatti + 1
WHERE (IDConc,IDProva) = (:concorrente,:prova);

CREATE OR REPLACE TRIGGER TENTATIVO_INVALIDO
BEFORE UPDATE ON Tentativi
REFERENCING OLD AS O
FOR EACH ROW
WHEN ((O.TentativiFatti >= ( SELECT NumMaxTentativi FROM  Prove
                                WHERE IDProva = O.IDProva))
 OR       (O.BestTime <= (SELECT TempoMax FROM  Prove
                            WHERE IDProva = O.IDProva)) )
SIGNAL SQLSTATE '70001' ('Prova superata o troppi tentativi!');
```

**Sistemi Informativi T**
**15 febbraio 2021**
**Risoluzione**

## 4) Progettazione logica (6 punti totali)

Dato lo schema concettuale in figura e considerando che:

a) le entità E1, E2 ed E3 vengono tradotte assieme;
b) nessuna associazione viene tradotta separatamente;




**4.1) [3 p.]** Si progettino gli opportuni schemi relazionali e si definiscano tali schemi mediante uno script SQL compatibile con DB2

```
-- il tipo degli attributi non è necessariamente INT
CREATE TABLE E1 (
K1          INT NOT NULL PRIMARY KEY,
A           INT NOT NULL,
K1R2        INT REFERENCES E1,
TIPO2       SMALLINT NOT NULL CHECK (TIPO2 in (1,2)),   -- se 2 l'istanza appartiene anche a E2
TIPO3       SMALLINT NOT NULL CHECK (TIPO3 in (1,3)),   -- se 3 l'istanza appartiene anche a E3
B           INT,
K1R1        INT REFERENCES E1,
C           INT,
CONSTRAINT TOTALE CHECK (TIPO2 = 2 OR TIPO3 = 3),
CONSTRAINT E2 CHECK (  (TIPO2 = 1 AND B IS NULL AND K1R1 IS NULL) OR
                       (TIPO2 = 2 AND B IS NOT NULL AND K1R1 IS NOT NULL)),
CONSTRAINT E3 CHECK (  (TIPO3 = 1 AND C IS NULL) OR (TIPO3 = 3 AND C IS NOT NULL))   );

CREATE TABLE R2D (      -- creata per rispettare la Prima Forma Normale
K1          INT NOT NULL REFERENCES E1,
D           INT NOT NULL,
PRIMARY KEY (K1,D)      );
```

**4.2) [3 p.]** Per i vincoli non esprimibili a livello di schema si predispongano opportuni trigger che evitino **inserimenti di singole tuple non corrette**

```
-- Trigger che garantisce che R2 referenzi un'istanza di E3
CREATE OR REPLACE TRIGGER R2_E3
BEFORE INSERT ON E1
REFERENCING NEW AS N
FOR EACH ROW
WHEN ( EXISTS ( SELECT *
                FROM    E1
                WHERE N.K1R2 = E1.K1
                AND     E1.TIPO3 = 1   ) )
SIGNAL SQLSTATE '70001' ('La tupla referenzia una tupla che non appartiene a E3!');

-- Trigger che impedisce di inserire valori di D se l'istanza di E1 non partecipa a R2
CREATE OR REPLACE TRIGGER D_VALIDI
BEFORE INSERT ON R2D
REFERENCING NEW AS N
FOR EACH ROW
WHEN ( NOT EXISTS ( SELECT *
                    FROM    E1
                    WHERE N.K1 = E1.K1
                    AND     E1.K1R2 IS NOT NULL ) )
SIGNAL SQLSTATE '70002' ('La tupla inserita riguarda un"istanza che non partecipa a R2! ');
```