Sistemi Informativi T
6 febbraio 2017
Risoluzione

Tempo a disposizione: 2:30 ore

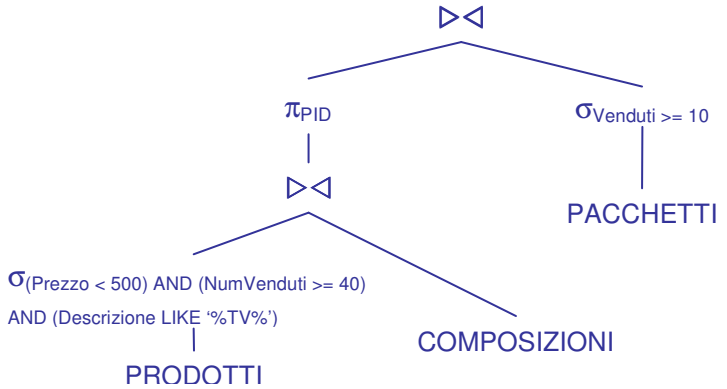
1) Algebra relazionale (3 punti totali):

Date le seguenti relazioni:

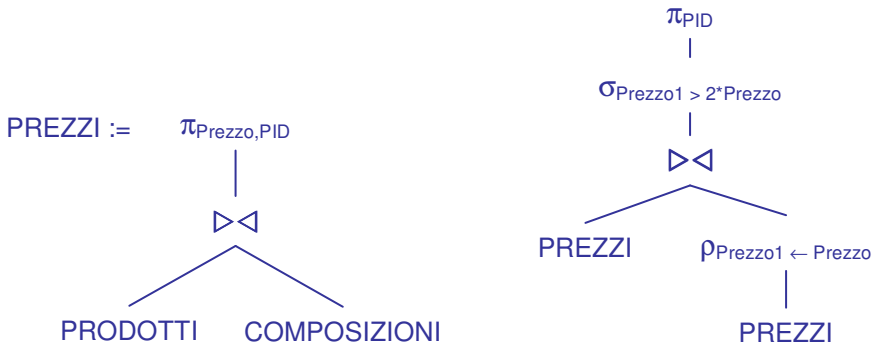
```
PRODOTTI(CodP,Descrizione,Prezzo,NumVenduti);
PACCHETTI(PID,Prezzo,Venduti);
COMPOSIZIONI(PID,CodP,NumUnita),
      PID REFERENCES PACCHETTI,
      CodP REFERENCES PRODOTTI;
--
-- Ogni pacchetto (o "bundle") consiste di uno o più prodotti,
-- ciascuno in quantità specificata da NumUnita.
-- Il Prezzo di un pacchetto è sempre inferiore a quello che si
-- pagherebbe comprando i vari prodotti separatamente.
-- NumVenduti e Venduti indicano quante unità di prodotto o
-- pacchetto, rispettivamente, sono state complessivamente vendute.
-- NB: NumVenduti tiene conto anche delle unità vendute all'interno
-- di pacchetti di cui il prodotto fa parte, oltre che delle unità
-- relative alle vendite del singolo prodotto.
-- Tutti i prezzi sono in formato DEC(6,2).
```

si scrivano in algebra relazionale le seguenti interrogazioni:

1.1) **[1 p.]** I dati dei pacchetti venduti almeno 10 volte in cui c'è almeno una 'TV' che costa meno di 500€ e complessivamente venduta 40 o più volte



1.2) **[2 p.]** I codici dei pacchetti che hanno almeno due prodotti in cui il prezzo singolo di uno è meno della metà dell'altro



Si noti che non è necessario mantenere nella vista PREZZI i codici dei prodotti, in quanto se i prezzi sono diversi lo sono necessariamente anche i relativi prodotti

Sistemi Informativi T
6 febbraio 2017
Risoluzione

SQL (5 punti totali)

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

2.1) **[2 p.]** Per ogni codice di prodotto, il numero di unità vendute singolarmente (ossia levando quelle vendute nei pacchetti). Il risultato va ordinato per valori decrescenti di questa quantità, escludendo i prodotti presenti in un solo pacchetto

```
SELECT   P.CodP, P.NumVenduti - SUM(C.NumUnita*B.Venduti) AS VenditeSingole
FROM     PRODOTTI P, PACCHETTI B, COMPOSIZIONI C
WHERE    B.PID = C.PID
AND      C.CodP = P.CodP
GROUP BY P.CodP, P.NumVenduti
HAVING   COUNT(*) > 1
   UNION
SELECT   P.CodP, P.NumVenduti AS VenditeSingole
FROM     PRODOTTI P LEFT JOIN COMPOSIZIONI C ON (C.CodP = P.CodP)
WHERE    C.PID IS NULL
ORDER BY VenditeSingole DESC
-- Il secondo operando di UNION seleziona i prodotti che non compaiono
-- in nessun pacchetto, che non vanno esclusi dal risultato
```

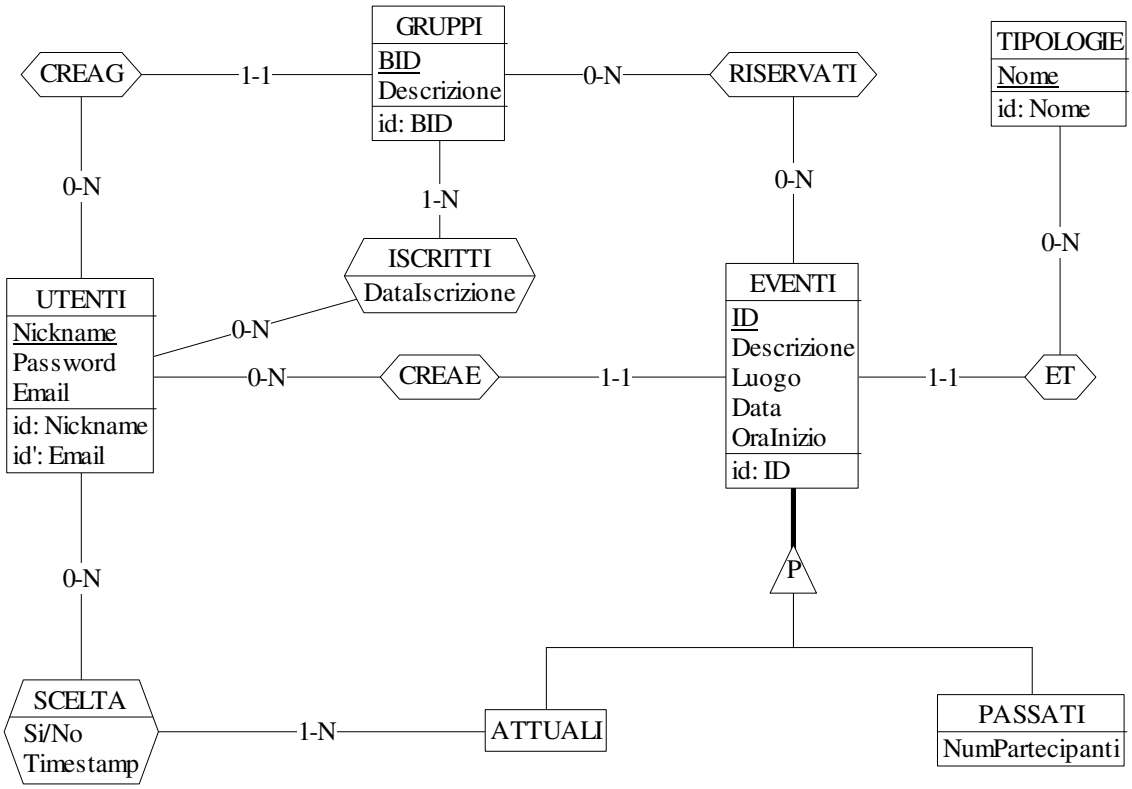
2.2) **[3 p.]** Il codice del prodotto per cui è minimo l'incasso totale di tutti pacchetti in cui il prodotto non è presente

```
-- Prima versione

WITH INCASSITOT (CodP, Totale) AS (
   SELECT P.CodP, SUM(B.Prezzo*B.Venduti)
   FROM   PACCHETTI B CROSS JOIN PRODOTTI P
   WHERE  B.PID NOT IN (SELECT C.PID
                        FROM COMPOSIZIONI C
                        WHERE C.Codp = P.CodP )
   GROUP BY P.CodP   )
SELECT   I.*
FROM     INCASSITOT I
WHERE    I.Totale = ( SELECT MIN(I1.Totale)
                      FROM   INCASSITOT I1  )

-- La c.t.e. calcola, per ogni prodotto, l'incasso totale dei pacchetti
-- in cui il prodotto non è presente. A tale scopo il blocco esterno
-- genera un prodotto Cartesiano, da cui vengono rimosse le coppie
-- (P.PCod,B.PID) che si trovano in COMPOSIZIONI. Ovviamente la clausola
-- FROM poteva anche scriversi: FROM PACCHETTI B, PRODOTTI P

-- Seconda versione: il prodotto per cui è minimo l' incasso totale
-- dei pacchetti in cui non c'è è evidentemente lo stesso che
-- massimizza il totale dei pacchetti in cui il prodotto è presente!

WITH INCASSITOT2 (CodP, Totale) AS (
   SELECT C.CodP, SUM(B.Prezzo*B.Venduti)
   FROM   PACCHETTI B, COMPOSIZIONI C
   WHERE  B.PID = C.PID
   GROUP BY C.CodP   )
SELECT   I.*
FROM     INCASSITOT2 I
WHERE    I.Totale = ( SELECT MAX(I1.Totale)
                      FROM   INCASSITOT2 I1  )
```

Sistemi Informativi T
6 febbraio 2017
Risoluzione

3) Progettazione concettuale (6 punti)

Il social network MeetUs è pensato per permettere ai suoi utenti di organizzare eventi di varia natura e condividerli con gli altri utenti del network. Ogni utente, identificato da un nickname, ha una password e un indirizzo di posta elettronica (lo stesso indirizzo non può essere usato per creare più utenti). Quando un utente crea un evento deve specificarne la tipologia (culturale, musicale, festa, ecc.), scegliendo tra le opzioni previste da MeetUs, il luogo, la data e l'orario di inizio. Può inoltre specificare se l'evento è riservato ai soli iscritti a certi gruppi. Un gruppo di MeetUs è creato da un utente, che ne diventa automaticamente membro. Per ogni iscritto a un gruppo si tiene traccia della relativa data di iscrizione.
Per ogni evento pubblicato, un utente può specificare se parteciperà o meno, e MeetUs registra il timestamp dell'eventuale scelta. Di un evento passato si archiviano solo le caratteristiche e il numero di utenti che hanno scelto di partecipare.



Commenti:

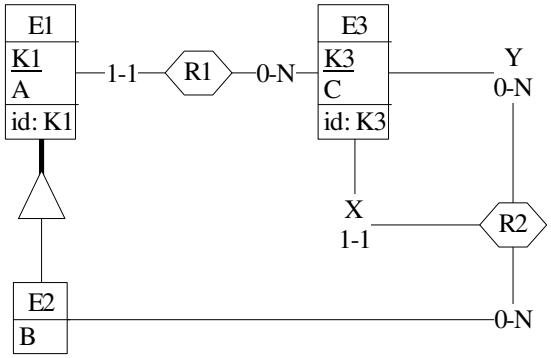
- Il vincolo che solo gli utenti appartenenti a GRUPPI cui è riservato un evento possono parteciparvi non è esprimibile in E/R.
- In ISCRITTI si mantiene anche il creatore del gruppo (vincolo di inclusione tra associazioni non esprimibile in E/R), da cui la cardinalità minima 1 con cui partecipa l'entità GRUPPI. In alternativa, il creatore si mantiene solo in CREAG e la cardinalità viene posta a 0. Analogamente per SCELTA, che include il creatore dell'evento (ovviamente con Si/No = 'Si').

Sistemi Informativi T
6 febbraio 2017
Risoluzione

4) Progettazione logica (6 punti totali)

Dato lo schema concettuale in figura e considerando che:

a) tutti gli attributi sono di tipo INT;
b) le entità E1 ed E2 vengono tradotte assieme;
c) le associazioni R1 e R2 non vengono tradotte separatamente;
d) un'istanza di E1 non può essere associata, tramite R1, a un'istanza di E3 che partecipa all'associazione R2 con ruolo Y;



4.1) [3 p.] Si progettino gli opportuni schemi relazionali e si definiscano tali schemi in DB2 (sul database SIT_STUD) mediante un file di script denominato SCHEMI.txt

```
CREATE TABLE E1 (
K1  INT NOT NULL PRIMARY KEY,
A   INT NOT NULL,
K3R1 INT NOT NULL,
TIPO2 SMALLINT NOT NULL CHECK (TIPO2 IN (1,2)),        -- 2 se istanza anche di E2
B   INT,
CONSTRAINT E2 CHECK (  (TIPO2 = 1 AND B IS NULL) OR  (TIPO2 = 2 AND B IS NOT NULL) )  );

CREATE TABLE E3 (
K3  INT NOT NULL PRIMARY KEY,
C   INT NOT NULL,
K3Y INT NOT NULL REFERENCES E3,
K1R2 INT NOT NULL REFERENCES E1        );

ALTER TABLE E1
ADD CONSTRAINT FK_E3_R1 FOREIGN KEY (K3R1) REFERENCES E3 ;
```

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**, definiti in un file TRIGGER.txt e usando se necessario il simbolo '@' per terminare gli statement SQL (altrimenti ';')

```
-- Trigger che garantisce che una tupla di E3 referenzi, tramite R2, una tupla di E2
CREATE TRIGGER R2_E2
BEFORE INSERT ON E3
REFERENCING NEW AS N
FOR EACH ROW
WHEN ( NOT EXISTS (        SELECT * FROM E1
                           WHERE N.K1R2 = E1.K1 AND E1.TIPO2 = 2                    ) )
SIGNAL SQLSTATE '70001' ('La tupla deve referenziare una tupla di E2!');

-- Il vincolo al punto d) impedisce che esistano valori  uguali nelle due foreign key E1.K3R1 e E3.K3Y.
-- Si hanno quindi 2 trigger simili, uno per l'inserimento in E1 e l'altro per l'inserimento in E3
CREATE TRIGGER PUNTO_D_E1
BEFORE INSERT ON E1
REFERENCING NEW AS N
FOR EACH ROW
WHEN ( EXISTS (     SELECT * FROM E3 WHERE N.K3R1 = E3.K3Y    ) )
SIGNAL SQLSTATE '70002' ('Inserimento non consentito!');

CREATE TRIGGER PUNTO_D_E3
BEFORE INSERT ON E3
REFERENCING NEW AS N
FOR EACH ROW
WHEN ( EXISTS (     SELECT * FROM E1 WHERE E1.K3R1 = N.K3Y    ) )
SIGNAL SQLSTATE '70003' ('Inserimento non consentito!');
```