Sistemi Informativi T
15 settembre 2014
Risoluzione

Tempo a disposizione: 2:30 ore

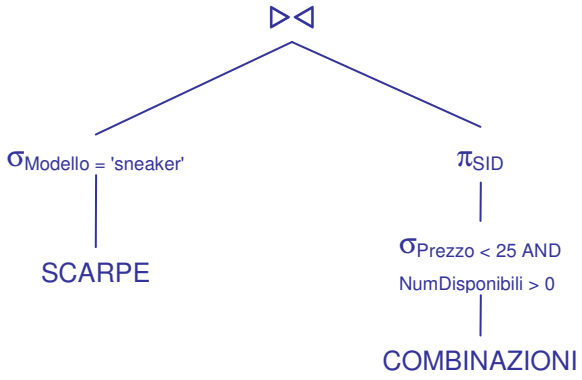
1) Algebra relazionale (3 punti totali):

Date le seguenti relazioni:

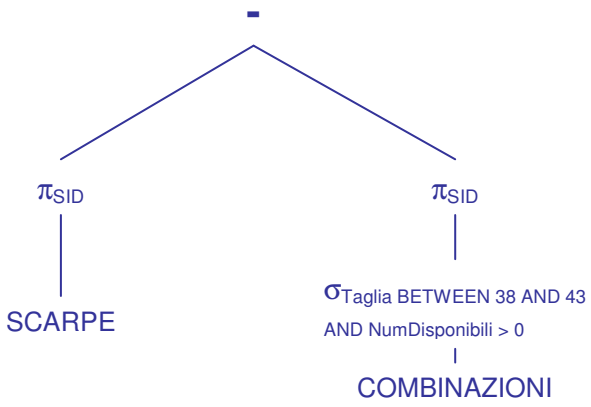
```
SCARPE(SID,Marca,Modello);
COMBINAZIONI(CID,SID,Taglia,Colore,Prezzo,NumDisponibili),
      SID REFERENCES SCARPE;
VENDITE(CID,Data,Numero),
      CID REFERENCES COMBINAZIONI;
--
-- Taglia ha valori interi positivi
-- NumDisponibili è un intero non negativo (0 = non disponibile)
-- Numero e' il numero di pezzi venduti in un dato giorno per una
-- data combinazione
```

si scrivano in algebra relazionale le seguenti interrogazioni:

1.1) [1 p.] I dati delle scarpe modello 'sneaker' che hanno almeno una combinazione disponibile a un prezzo minore di 25€



1.2) [2 p.] I codici delle scarpe non disponibili in nessun colore per nessuna taglia compresa tra 38 e 43



Sistemi Informativi T
15 settembre 2014
Risoluzione

2) SQL (5 punti totali)

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

2.1) [2 p.] Per ogni combinazione, l'incasso totale del 2014

```
SELECT   C.CID, C.Prezzo*SUM(V.Numero) AS Totale2014
FROM     COMBINAZIONI C, VENDITE V
WHERE    C.CID = V.CID
AND      YEAR(V.Data) = 2014
GROUP BY C.CID
```

2.2) [3 p.] Per ogni scarpa di modello 'sneaker', la combinazione (taglia e colore) più venduta nel 2014

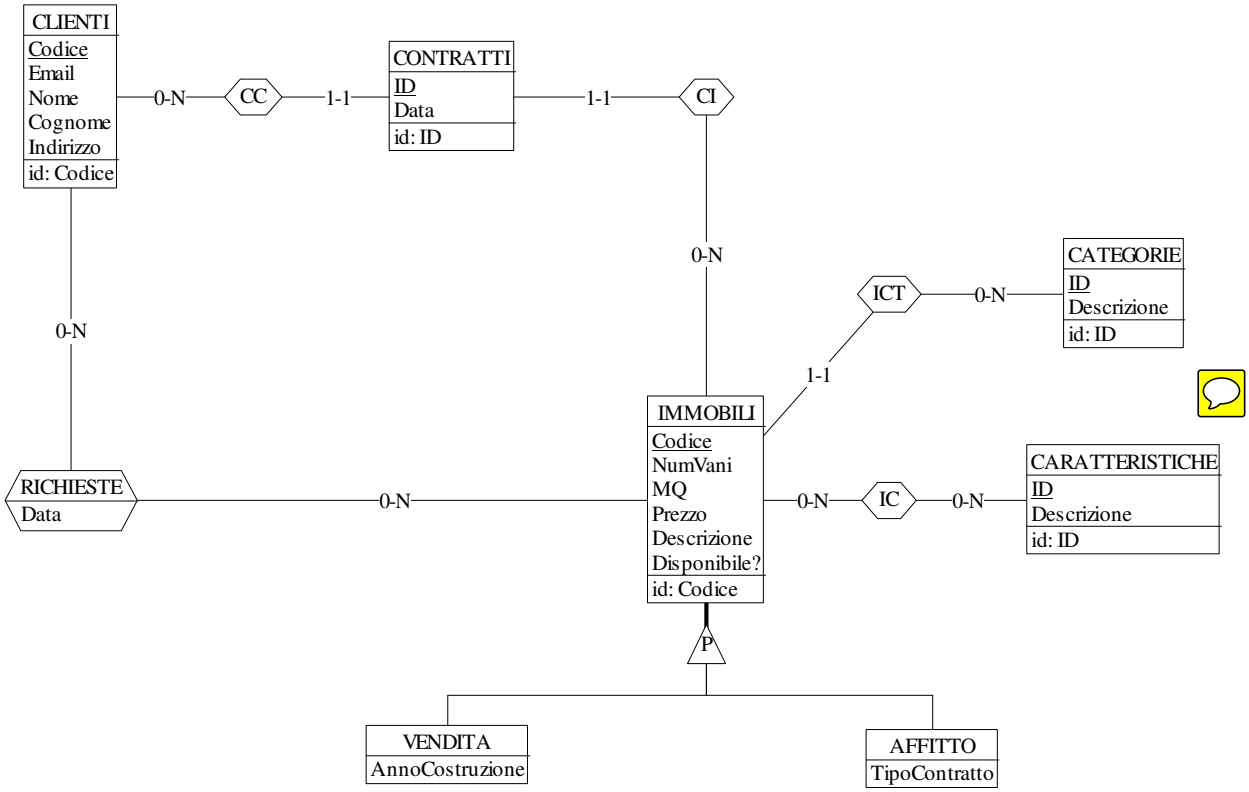
```
WITH TOTSNEAKERS (CID,Taglia,Colore,SID,TotVendite) AS (
   SELECT   C.CID, C.Taglia, C.Colore, S.SID, SUM(V.Numero)
   FROM     SCARPE S, COMBINAZIONI C, VENDITE V
   WHERE    S.SID = C.SID
   AND      C.CID = V.CID
   AND      S.Modello = 'sneaker'
   AND      YEAR(V.Data)  = 2014
   GROUP BY C.CID, C.Taglia, C.Colore, S.SID       )                )

SELECT   T.*
FROM     TOTSNEAKERS T
WHERE    T.TotVendite = (  SELECT MAX(T1.TotVendite)
                           FROM   TOTSNEAKERS T1
                           WHERE  T1.SID = T.SID )
```

Sistemi Informativi T
15 settembre 2014
Risoluzione

3) Progettazione concettuale (6 punti)

L'agenzia di intermediazione @HOME tratta l'affitto e la vendita di immobili di diverse categorie predefinite (casa colonica, appartamento, villa, ecc.). Ogni immobile è caratterizzato da un codice univoco, dal numero di vani, metri quadri, prezzo e da una descrizione testuale. Inoltre, per ogni caratteristica booleana gestita da @HOME viene detto se l'immobile la possiede o meno (ad es.: giardino, garage, posto auto, ascensore). Per gli immobili in affitto si mantiene il tipo di contratto stipulabile, mentre per quelli in vendita l'anno di costruzione.
L'agenzia registra i suoi clienti identificandoli tramite un codice e riportandone i dati anagrafici.
I clienti possono fare richieste di affitto/acquisto e stipulare contratti. Di ogni richiesta si memorizza la data e l'immobile interessato. Sia che il cliente in seguito si dimostri non più interessato, sia che concluda il relativo contratto, la richiesta viene rimossa dal database. Di ogni contratto stipulato la @HOME mantiene il tipo (affitto o vendita), il cliente e l'immobile interessato. Quando si stipula un contratto, l'immobile relativo risulta non più disponibile, ma può tornarlo se viene rimesso in vendita o termina il periodo di affitto.
NB: Per semplicità, se lo stesso immobile viene, ad esempio, prima venduto e poi messo in affitto, la @HOME lo gestisce come se fossero immobili differenti.



Commenti:

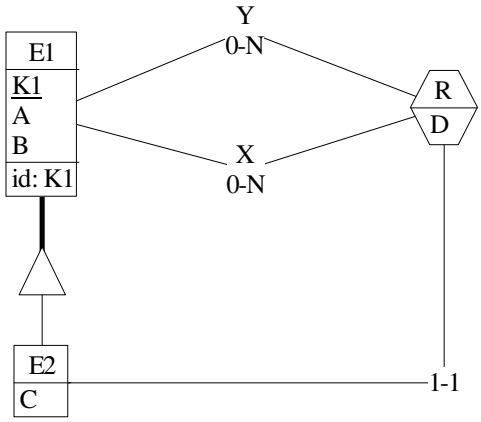
1. L'unico elemento degno di commento riguarda l'entità IMMOBILI, che nella soluzione proposta non viene partizionata in immobili disponibili o meno (il che ovviamente impedisce di rappresentare il vincolo che una richiesta può riferirsi solo a un immobile disponibile). Per contro va considerato che la distinzione dovuta alla disponibilità non è rilevante per i CONTRATTI, in quanto un contratto può anche riguardare un immobile che è tornato a essere disponibile. Inoltre, la partizione di IMMOBILI sulla base delle disponibilità (oltre a quella già presente) è problematica in DB-Main, che non consente a un'entità di essere radice di due gerarchie distinte.
2. Si noti in oltre che, con l'ipotesi semplificativa descritta nel "Nota Bene", ogni contratto è univocamente individuato come di affitto o vendita.

Sistemi Informativi T
15 settembre 2014
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 separatamente;
c) l'associazione R non viene tradotta separatamente;
d) le istanze di E1 associate tramite i ruoli X e Y sono tali per cui il valore di A della prima (X) è sempre almeno il doppio del valore di B della seconda (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,
B INT NOT NULL,
TIPO SMALLINT NOT NULL CHECK (TIPO IN (1,2)),          -- 2: istanza anche di E2
);

CREATE TABLE E2 (
K1 INT NOT NULL PRIMARY KEY REFERENCES E1,
C INT NOT NULL,
D INT NOT NULL,
K1X INT NOT NULL REFERENCES E1,
K1Y INT NOT NULL REFERENCES E1);
```

4.2) [3 p.] Per i vincoli non esprimibili a livello di schema si predispongano opportuni **trigger che evitino inserimenti di tuple non corrette**, definiti in un file TRIGGER.txt e usando se necessario il simbolo '@' per terminare gli statement SQL (altrimenti ';')

```
-- Trigger che garantisce il rispetto del vincolo di cui al punto d).

CREATE TRIGGER PUNTO_D
BEFORE INSERT ON E2
REFERENCING NEW AS N
FOR EACH ROW
WHEN (EXISTS ( SELECT * FROM E1 E1X, E1 E1Y
                    WHERE   N.K1X = E1X.K1
                    AND     N.K1Y = E1Y.K1
                    AND     E1X.A < 2*E1Y.B          ) )
SIGNAL SQLSTATE '70001' ('Le tuple referenziate non rispettano il vincolo su A e B!');
```