Sottoprova di SQL

Le seguenti relazioni definiscono una base di dati "**Prenotazioni**" per gestire le prenotazioni di camere in un hotel di Torino. Gli attributi sottolineati sono le chiavi primarie delle relazioni.

CLIENTE(NumTessera, Nome, DataNascita, CittàResidenza)
CAMERA(Num, Piano, NumPostiLetto, Tipo, Prezzo)
PRENOTA(Cliente, Num, Piano, DataArrivo, DataPartenza)

Vincoli di integrità referenziali: PRENOTA(Cliente) referenzia CLIENTE(NumTessera) e PRENOTA(Num,Piano) referenzia CAMERA(Num,Piano).
Significato degli attributi: "NumTessera" = numero tessera del cliente accreditato; "Tipo" può assumere i valori "economica", "normale" e "suite"; "Prezzo" è il costo di una notte in euro. I rimanenti attributi sono autoesplicativi.
Gli attributi sono tutti NOT NULL.

Domanda 1 (bassa complessità).
Con riferimento alla base dati "Prenotazioni" esprimere in SQL la seguente interrogazione:
Trovare i clienti che hanno varie prenotazioni di camere con un numero diverso di posti letto.

Soluzione 1.

```
SELECT DISTINCT p1.Cliente
FROM prenota p1 JOIN prenota p2 ON (p1.Cliente=p2.Cliente)
     JOIN camera c1 ON (p1.Num=c1.Num AND p1.Piano=c1.Piano)
     JOIN camera c2 ON (p2.Num=c2.Num AND p2.Piano=c2.Piano)
WHERE c1.NumPostiLetto<>c2.NumPostiLetto;
```

Domanda 2 (media complessità).
Con riferimento alla base dati "Prenotazioni" esprimere in SQL la seguente interrogazione:
Mostrare i clienti che hanno pagato, per una singola notte, un prezzo maggiore del prezzo medio pagato, per una singola notte, dai clienti provenienti dalla loro stessa città.

Soluzione 2.

```
SELECT DISTINCT cl.NumTessera, cl.Nome, cl.CittàResidenza
FROM cliente cl JOIN prenota p ON (cl.NumTessera=p.Cliente)
     JOIN camera ca ON (p.Num=ca.Num AND p.Piano=ca.Piano)
WHERE ca.Prezzo > (SELECT AVG(Prezzo)
                   FROM cliente cl1 JOIN prenota p1 ON (cl1.NumTessera=p1.Cliente)
                        JOIN camera ca1 ON (c1.Num=p1.Num AND c1.Piano=1.Piano)
                    WHERE cl1.CittàResidenza=cl.CittàResidenza);
```

Domanda 3 (alta complessità).
Con riferimento alla base dati "Prenotazioni" esprimere in SQL la seguente interrogazione:
Trovare, SENZA utilizzare gli operatori insiemistici, quanti clienti hanno sempre e soltanto prenotato camere di tipo economico.

Soluzione 3.

```
SELECT COUNT(DISTINCT Cliente)
FROM prenota
WHERE Cliente<>ALL
              (SELECT p.Cliente
               FROM prenota p JOIN camera c ON (p.Num=c.Num AND p.Piano=c.Piano)
               WHERE c.Tipo<>'economica');
```

Sottoprova di Algebra, calcolo relazionale e ottimizzazione logica

Domanda 1.
Con riferimento alla base di dati "**Prenotazioni**":

A. Esprimere in Algebra Relazionale l'interrogazione
Elencare i clienti che hanno prenotato almeno una suite in ogni piano dell'hotel.

B. Esprimere, nel calcolo dei predicati su tuple con dichiarazione di range, la seguente domanda:
Elencare le camere in overbooking con un solo posto letto (si richiede il Num, Piano). (VERSIONE 1: Per semplicità consideriamo in overbooking una camera assegnata a clienti diversi con stessa data di arrivo) (VERSIONE 2: Una camera in overbooking è una camera assegnata a clienti diversi per periodi che coincidono anche solo in parte).

Soluzione 1.

A. Una possibile soluzione (in algebra relazionale) è la seguente (con il quoziente):

$$R(A,B) \div S(B)$$

dove

$$R(A,B) = \pi_{Cliente,Piano}(\sigma_{Tipo='suite'}(PRENOTA \bowtie CAMERA))$$

$$S(B) = \pi_{Piano}(\sigma_{Tipo='suite'}(CAMERA))$$

B. Una possibile soluzione è la seguente:

VERSIONE 1: {p.Num, p.Piano | p(PRENOTA) | ∃c(CAMERA)(c.NumPostiLetto=1 ∧ p.Num=c.Num ∧ p.Piano=c.Piano ∧ ∃p'(PRENOTA) (p.Num=p'.Num ∧ p.Piano=p'.Piano ∧ p.Cliente≠p'.Cliente ∧ p.DataArrivo=p'.DataArrivo))}

VERSIONE 2: {{p.Num, p.Piano | p(PRENOTA) | ∃c(CAMERA)(c.NumPostiLetto=1 ∧ p.Num=c.Num ∧ p.Piano=c.Piano ∧ ∃p'(PRENOTA) (p.Num=p'.Num ∧ p.Piano=p'.Piano ∧ p.Cliente≠p'.Cliente ∧ (p.DataArrivo ≤ p'.DataArrivo < p.DataPartenza ∨ p'.DataArrivo ≤ p.DataArrivo < p'.DataPartenza)}

Domanda 2.

Data la seguente query sulla base di dati "**Prenotazioni**":

$$\sigma_{Nome='Jack\ Torrance' \wedge Tipo='Suite'}((cliente \bowtie_{numTessera=Cliente} prenota) \bowtie camera)$$

disegnare gli alberi sintattici prima e dopo l'ottimizzazione logica. Per semplicità **non** considerare le varianti di parsificazione date dalla proprietà associativa dei join.
Inoltre calcolare il numero di tuple "mosse" prima e dopo l'ottimizzazione logica. Si svolgano i calcoli sapendo che:

CARD(prenota) = 60 000
CARD(cliente) = VAL(Cliente,prenota) = 600
CARD(camera) = 30
VAL(Nome,cliente) = 300

Soluzione 2.
Decomponendo l'AND della selezione e trasferendo le selezioni verso le foglie (passi 1 e 2 dell'algoritmo di ottimizzazione logica) ottengo:

$$\left(\sigma_{Nome='Jack\ Torrance'}(cliente) \bowtie_{numTessera=Cliente} prenota\right) \bowtie \sigma_{Tipo='Suite'}(camera)$$

Gli alberi sintattici possono essere facilmente ricavati.

Prima dell'ottimizzazione.

Chiamo r il primo join $r = cliente \bowtie_{numTessera=Cliente} prenota$

r combina CARD(cliente)=600 tuple con CARD(prenota)=60000 tuple, cioè $600 \cdot 60000 = 36\ milioni$ di tuple per produrre $|r|$ tuple, dove

$$|r| = \min\left\{\frac{1}{VAL(NumTessera, cliente)}, \frac{1}{VAL(Cliente, prenota)}\right\} \cdot CARD(cliente) \cdot CARD(prenota)$$

Dato che NumTessera è chiave primaria di cliente, $VAL(NumTessera, cliente) = CARD(Cliente)$,

$$|r| = \min\left\{\frac{1}{600}, \frac{1}{600}\right\} \cdot 600 \cdot 60000 = 60000$$

Il secondo join combina $|r| = 60000$ tuple e $CARD(camera) = 30$ tuple, cioè $60000 \cdot 30 = 1{,}8\ milioni$ di tuple per produrre

$|r \bowtie camera| = \min\left\{\frac{1}{VAL(\langle Num,Piano\rangle, r)}, \frac{1}{VAL(\langle Num,Piano\rangle, camera)}\right\} \cdot CARD(r) \cdot CARD(camera)$.

Dato che:

- $\langle Num, Piano\rangle$ è la chiave primaria di Camera e quindi $VAL(\langle Num, Piano\rangle, camera) = |camera| = 30$
- $VAL(\langle Num, Piano\rangle, r) = 30$

$|r \bowtie camera| = \min\left\{\frac{1}{30}, \frac{1}{30}\right\} \cdot 60000 \cdot 30 = 60000$.

La selezione considera $|r \bowtie camera| = 60000$ tuple.

Quindi "muovo":

- Primo join: 36 milioni di tuple
- Secondo join: 1,8 milioni di tuple
- Selezione: 60000 tuple

cioè quasi 40 milioni di tuple.

Dopo l'ottimizzazione.

$$\left(\sigma_{Nome='Jack\ Torrance'}(cliente) \bowtie_{numTessera=Cliente} prenota\right) \bowtie \sigma_{Tipo='Suite'}(camera)$$

La prima selezione considera CARD(cliente)=600 tuple per ottenere $\left|\sigma_{Nome='Jack\ Torrance'}(cliente)\right| = \frac{1}{300} \cdot 600 = 2$ tuple.

Chiamo r' il primo join $r' = \sigma_{Nome='Jack\ Torrance'}(cliente) \bowtie_{numTessera=Cliente} prenota$.

r' combina $\left|\sigma_{Nome='Jack\ Torrance'}(cliente)\right| = 2$ tuple e CARD(prenota)=60000 tuple, cioè $2 \cdot 60000 = 120\ mila$ tuple per produrre $|r'|$ tuple, dove

$$|r'| = \min\left\{\frac{1}{VAL\left(NumTessera, \sigma_{Nome='Jack\ Torrance'}(cliente)\right)}, \frac{1}{VAL(Cliente, prenota)}\right\} \cdot$$

$$CARD\left(\sigma_{Nome='Jack\ Torrance'}(cliente)\right) \cdot CARD(prenota).$$

Dato che NumTessera è la chiave primaria di cliente, $VAL\left(NumTessera, \sigma_{Nome='Jack\ Torrance'}(cliente)\right) = CARD\left(\sigma_{Nome='Jack\ Torrance'}(cliente)\right) = 2$, quindi

$|r'| = \min\left\{\frac{1}{2}, \frac{1}{600}\right\} \cdot 2 \cdot 60000 = \frac{1}{600} \cdot 2 \cdot 60000 = 2 \cdot 100 = 200$ tuple.

Dato che ci sono tre tipi di camere, VAL(Tipo,camera)=3 e $\sigma_{Tipo='Suite'}(camera)$ considera CARD(camera) = 30 tuple per produrre $VAL(Tipo, camera) \cdot CARD(camera) = \frac{1}{3} \cdot 30 = 10$ tuple.

Il secondo join combina $|r'| = 200$ tuple e $CARD\left(\sigma_{Tipo='Suite'}(camera)\right) = 10$ tuple, cioè $200 \cdot 10 = 2000$ tuple.

Quindi "muovo":

- Primo selezione: 600 tuple
- Primo join: 120 mila tuple
- Seconda selezione: 30 tuple
- Secondo join: 2 mila tuple

cioè quasi 123 mila tuple.

Sottoprova di Teoria

Domanda 1.
Dati: R(L, M, N, O, P, Q, R) e F = { O → M R, N → L M Q, M N → O P, R → N O }
Dire, motivando la risposta, se R è in 3FN e se non lo è decomporla in relazioni in 3FN esplicitando tutti i passaggi.

Soluzione 1.
Per prima cosa è necessario identificare le chiavi candidate. In questo caso abbiamo tre chiavi K1={O}, K2={N}, K3={R} (il calcolo della chiusura lo dimostra).

La relazione R è già in 3FN, in quanto tutte le dipendenze funzionali sono del tipo "superchiave". Non è necessario decomporre.

Domanda 2.
A. Spiegare in modo succinto ma preciso le principali differenze tra B-tree e B+-tree e quali vantaggi apportano i B+-tree rispetto ai B-tree nella gestione degli indici. Attenzione: non si richiedono le definizioni, ma le differenze e i vantaggi, quindi una rielaborazione di ciò che è stato spiegato.
B. Disegnare un B+-tree con m=4 contenente come chiavi i numeri da 1 a 20. Non occorre simulare gli inserimenti (le chiavi potrebbero essere state inserite in un ordine qualunque) ma soltanto rappresentare un possibile B+-tree contenente tutte le chiavi da 1 a 20.

Soluzione 2.
A. Le differenze e i vantaggi possono essere tratti dal libro di testo o dalle slide.
B. È valido qualsiasi B+-tree che rispetta le proprietà dei B-tree e dei B+-tree riportate sul libro di testo e sulle slide.

Domanda 3.
Considerare la seguente storia interfogliata S = r1(x), r1(y), r2(x), r3(y), w1(y), r2(y), w2(x)
La storia S è compatibile con il protocollo 2PL? Giustificare la risposta.

Soluzione 3.
La storia S è compatibile con 2PL perché esiste almeno un modo di aggiungere i lock (lock condivisi o esclusivi per le letture e lock esclusivi per le scritture) rispettando il 2PL (cioè nessuna transazione acquisisce lock dopo avere rilasciato un lock precedentemente acquisito). Un esempio di aggiunta dei lock 2PL è il seguente:

S = LS1(x), r1(x), LS1(y), r1(y), LS2(x), r2(x), LS3(y), r3(y), UN3(y), LX1(y), w1(y), UN1(x), UN1(y), LS2(y), r2(y), LX2(x), w2(x), UN2(y), UN2(x)

Domanda 4.
Si consideri un file di log L con il seguente contenuto in seguito ad un crash:

```
<T1,START>;
<T2,START>;
<T1,BS(t1[A],5),AS(t1[A],10)>;
<T2,BS(t2[B],3),AS(t2[B],5)>;
<T3, START>;
<T2,COMMIT>;
-- checkpoint --;
```

<T3,BS(t3[C],3),AS(t3[C],5)>;
<T3,ABORT>

crash!

Indicare il contenuto del record di checkpoint.

Soluzione 4.
Il record di checkpoint conterrà la lista delle transazioni non terminate (quindi T1 e T3) con i relativi puntatori ai rispettivi <Ti, START>. Contiene inoltre l'OK riguardante la buona riuscita delle operazioni previste dal checkpoint, quindi: `[(T1,p1);(T3,p3);OK].`