Corso di Basi di Dati 1

Quarto Compitino - Soluzioni

22 Gennaio 2003

1. $NomeUt, IndUt, TelUt, TipoUt, NumL, GenereL, TitoloL, AutoreL, Durata, DataP, DataR$

   $NomeUt \rightarrow IndUt \;\; TelUt \;\; TipoUt$
   $NumL \rightarrow TitoloL \;\; AutoreL \;\; GenereL$
   $TipoUt \;\; GenereL \rightarrow Durata$
   $NumL \;\; DataP \rightarrow NomeUt \;\; DataR$
   $NumL \;\; DataR \rightarrow NomeUt \;\; DataP$

   (a) • relazione su tale schema che soddisfa le dipendenze: $r = \emptyset$
       • relazione su tale schema che non soddisfa le dipendenze:

NomeUt	IndUt	TelUt	TipoUt	NumL	GenereL	TitoloL	AutoreL	Durata	DataP	DataR
mario rossi	genova	...	...	...	...	...	...	...	...	...
mario rossi	milano	...	...	...	...	...	...	...	...	...

   (b) chiavi $(NumL, DataP)$ e $(NumL, DataR)$

   (c) lo schema non è in BCNF: ad es. $TipoUt \;\; GenereL \rightarrow Durata$ non contiene a sinistra una chiave

   (d) lo schema non è in 3NF: ad es. $TipoUt \;\; GenereL \rightarrow Durata$ non contiene a sinistra una chiave e l'attributo a destra non é primo

   (e) decomposizione lossless join dello schema in BCNF:
   - applicando l'algoritmo, considerando le dipendenze nell'ordine dato si ottiene:
     $(NomeUt, IndUt, TelUt, TipoUt)$, con chiave $NomeUt$
     $(NumL, TitoloL, AutoreL, GenereL)$, con chiave $NumL$
     $(NomeUt, NumL, Durata)$, con chiave $(NomeUt, NumL)$
     $(NumL, DataP, NomeUt, DataR)$, con chiavi $(NumL, DataP)$ e $(NumL, DataR)$
     la dipendenza $TipoUt \;\; GenereL \rightarrow Durata$ non é preservata
   - applicando l'algoritmo, considerando per prima la terza dipendenza e dopo le prime due si ottiene la decomposizione in (f), che preserva le dipendenze

   (f) decomposizione lossless join che preservi le dipendenze dello schema in 3NF:
   le dipendenze non sono minimali, oltre a piú di un attributo a destra, la dipendenza $NumL \;\; DataR \rightarrow NomeUt$ é ridondante
   applicando l'algoritmo si ottiene:
   $(NomeUt, IndUt, TelUt, TipoUt)$, con chiave $NomeUt$
   $(NumL, TitoloL, AutoreL, GenereL)$, con chiave $NumL$
   $(TipoUt, GenereL, Durata)$, con chiave $(TipoUt, GenereL)$
   $(NumL, DataP, NomeUt, DataR)$, con chiavi $(NumL, DataP)$ e $(NumL, DataR)$
   che é anche in BCNF.

2. (a) tutte e tre le alternative viste (SQL esteso, interfacce per DBMS, SQL embedded) sono utilizzabili;
   (b) schema (es. in PL/SQL):
   - procedura con parametro `ndip` di tipo `IMPIEGATI.Dip#%TYPE`
   - dichiaro `c` cursore per
     ```
     SELECT Nome, Mansione, Stipendio
     FROM IMPIEGATI
     WHERE Dip# = ndip
     ```
   - scrivo su file html che dovrá contenere l'output l'intestazione e apro la lista con < ul >
   - apro il cursore con `open c`
   - faccio un ciclo loop con uscita quando `c%NOTFOUND` in cui:
     - assegno alle opportune variabili il contenuto della tupla correntemente puntata dal cursore (con `FETCH INTO`)
     - scrivo nel file un item con < li > e il contenuto di tali variabili
   - chiudo il cursore
   - scrivo nel file html la chiusura della lista con < \ul >
   - scrivo nel file html i tag di chiusura del file

3. (a) indice su $Imp\#$ con 15 entrate:

7369	p_7
7499	p_{13}
...	...
7934	p_4

   (b) non ha senso: la relazione non é clusterizzata sulla chiave primaria di IMPIEGATI
   (c) non ha senso: la relazione non é clusterizzata sulla chiave primaria di IMPIEGATI
   (d) non ha senso: un indice sparso non clusterizzato non aiuta a ridurre i costi di ricerca
   (e) indice su $Mansione$ con 4 entrate:

dirigente	p_5, p_8, p_{16}
ingegnere	$p_2, p_3, p_4, p_7, p_{10}, p_{18}$
segretaria	p_9, p_{11}, p_{15}
tecnico	p_{13}, p_{14}, p_{17}

   (f) indice su $Dip\#$ con 2 entrate:

10	p_2
30	p_{13}

   (g) indice su $Dip\#$ con 3 entrate:

10	p_2
20	p_7
30	p_{13}

   (h) non ha senso: un indice sparso non clusterizzato non aiuta a ridurre i costi di ricerca

4.
```
SELECT GIOCATORE.NTessera, COUNT(*)
FROM GIOCATORE, GOAL
WHERE GIOCATORE.NTessera = GOAL.NTessera
      AND Ruolo = 'attaccante'
      AND AutoGoal = FALSE AND Minuto > 80
```

(a) numero di tuple di GIOCATORE e GOAL, numero di valori distinti per Ruolo, intervallo di valori per Minuto

(b) clustering di GIOCATORE e GOAL su NTessera

(c) su Ruolo per GIOCATORE, su Minuto per GOAL (ad albero, non hash), su AutoGoal per GOAL

inoltre se relazioni non clusterizzate su NTessera, un indice su una delle due su NTessera aiuterebbe nell'esecuzione del join

(d) nessuno, altri indici sono inutili ma non dannosi

5. (a) *Controllo della concorrenza:* consideriamo il seguente schedule delle due transazioni

T_1	T_2
$read(X)$	
$X := X - 50$	
	$read(Y)$
	$Y := Y - 100$
	$write(Y)$
	$read(X)$
	$X := X + 100$
	$write(X)$
	$commit$
$write(X)$	
$read(Z)$	
$Z := Z + 50$	
$write(Z)$	
$commit$	

entrambe le transazioni leggono il valore originale di X (100), e la modifica effettuata da T_2 é persa (lost update), perché il valore finale di X sará quello scritto da T_1, cioé 50. Il problema é dovuto all'ordine in cui le due transazioni effettuano le operazioni in conflitto su X, che non sarebbe realizzabile con un protocollo di lock opportuno (es. 2PL).

(b) *Ripristino:* consideriamo la seguente esecuzione (parziale) della transazione T_1:

T_1
$read(X)$
$X := X - 50$
$write(X)$
$read(Z)$
$Z := Z + 50$
$crash$

se non si utilizzano meccanismi di logging, al momento del crash la base di dati conterrá $X = 50$ e $Z = 200$. A questo punto né non rieseguendo, né rieseguendo T_1 si potrá ottenere l'effetto voluto (cioé trasferire 50 da X a Z, che corrisponde allo stato $X = 50, Z = 250$). Il problema é che la scrittura del valore di X é stata effettuata prima di essere sicuri che la transazione completasse con successo e senza registrare le informazioni necessarie per effettuare l'undo della transazione in caso di abort.

(c) *Isolamento:* la proprietá di isolamento richiede che gli effetti delle transazioni che non hanno ancora eseguito il commit non siano visibili ad altre transazioni. Consideriamo la seguente esecuzione (parziale) delle due transazioni:

T_1	T_2
$read(X)$	
$X := X - 50$	
$write(X)$	
	$read(Y)$
	$Y := Y - 100$
	$write(Y)$
	$read(X)$
	$X := X + 100$
	$write(X)$
	$commit$
$read(Z)$	
$Z := Z + 50$	
$abort$	

Poiché T_2 legge il valore di X modificato da T_1 (50), e poi T_1 viene abortita, si ottiene lo stato finale (non consistente, cioé in cui é violato il vincolo $X + Y + Z = 800$) $X = 150$, $Y = 400$, $Z = 200$, a meno che non si esegua un rollback a cascata di T_2.