Basi di dati II — 25 febbraio 2014

Tempo a disposizione: due ore.

Cognome _____ **Nome** _____ **Matricola** _____ **Ordin.** _____

Domanda 1 (20%)

Nella figura seguente è schematizzato un piccolissimo buffer con quattro pagine (numerate da 0 a 3), il cui stato viene descritto, per ciascuna pagina, da (i) un intero che indica il numero di pin su di essa (quindi 0 indica che la pagina è libera) (ii) un riferimento al blocco che per ultimo è stato caricato nella pagina; (iii) l'istante in cui è stato effettuato l'ultimo caricamento; (iv) l'istante in cui la pagina è stata per l'ultima volta liberata (se è libera) (v) un booleano che indica se la pagina è sporca (1 indica che è stata modificata dopo il caricamento o dopo l'ultima scrittura).

Pagina del buffer:	0	1	2	3
numero di pin sulla pagina	1	2	1	0
blocco	62	33	47	35
istante load	1	7	9	3
istante unpin				8
dirty	1	0	0	0

Si supponga ora che vengano eseguite (a partire dall'istante 10, dopo che all'istante 9 è stato caricato il blocco 47 nella pagina 2) le seguenti operazioni (in cui l'argomento delle pin e unpin è il blocco di interesse, con setXXX si indica un aggiornamento di un qualche valore nel blocco e con flushAll il flush dell'intero buffer):

Istante	10	11	12	13	14	15
Operazione	unpin(62),	pin(56),	setXXX(56,...)	unpin(56),	flushAll	setXXX(47,...)

Istante	16	17	18	19	20	21	22
Operazione	unpin(47)	pin(62),	setXXX(62,...)	unpin(62),	pin(56),	unpin(56),	pin(62)

Riempiendo la tabella seguente, indicare, per ciascuna delle strategie e per ciascuna delle pin, (i) quale pagina del buffer viene utilizzata, (ii) se il blocco corrispondente viene letto dal disco e (iii) se viene eseguita una scrittura su disco.

		naif			LRU			clock		
Istante	Operazione	Pagina	Legge?	Scrive?	Pagina	Legge?	Scrive?	Pagina	Legge?	Scrive?
11	pin(56)									
17	pin(62)									
20	pin(56)									
22	pin(62)									

Basi di dati II — 25 febbraio 2014

Domanda 2 (20%)
Indicare il risultato di ciascuna delle seguenti interrogazioni XQuery:

```
let $i:= (3,2,1)
for $j in (1,2,3)
return <result>{$i} {$j}</result>
```

```
for $i in (1,2,3)
let $j:= (3,2,1)
return <result>{$i} {$j}</result>
```

```
for $i in (1,2,3)
for $j in (3,2,1)
return <result>{$i} {$j}</result>
```

```
let $i:= (1,2,3)
let $j:= (3,2,1)
return <result>{$i} {$j}</result>
```

Basi di dati II — 25 febbraio 2014

Domanda 3 (20%) Ai fini dello studio congiunto di affidabilità e concorrenza, è necessario considerare negli schedule anche le operazioni di commit e abort (c_i e a_i indicano rispettivamente il commit e l'abort della transazione T_i) e considerare come schedule anche i relativi prefissi (cioè porzioni di schedule eseguite fino ad un certo momento, costituite anche da transazioni non concluse). Allo scopo, si considerino le seguenti definizioni, in cui per "schedule" si intende più in generale un "prefisso di schedule" e per "coppia di transazioni T_i, T_j in s," si intende "coppia di transazioni T_i, T_j distinte (cioÃ¨ con $i \neq j$) che compaiano, almeno con un prefisso non vuoto, in s"; inoltre, si assuma che se T_i legge x da T_j, l'abort a_j non compare fra la scrittura $w_j(x)$ e la lettura $r_i(x)$:

- uno schedule s è *recoverable (RC)* se, per ogni coppia di transazioni T_i, T_j in s, vale la proprietà seguente: se T_i legge un qualche x da T_j in s e il commit c_i di T_i compare in s, allora anche il commit c_j di T_j compare in s e lo precede (in simboli $c_j <_s c_i$)
- uno schedule *evita gli abort in cascata (EAC)* se, per ogni coppia di transazioni T_i, T_j, vale la proprietà seguente: se T_i legge x da T_j in s allora $c_j <_s r_i(x)$ (il commit c_j di T_j compare nello schedule e precede la lettura di x da parte di T_i)
- uno schedule *è stretto (ST)* se, per ogni coppia di transazioni T_i, T_j, vale la proprietà seguente: se ci sono un'operazione $o_i(x)$ (lettura o scrittura) e una $w_j(x)$ con $w_j(x) <_s o_i(x)$, allora s deve includere anche c_j con $c_j <_s o_i(x)$ oppure a_j con $a_j <_s o_i(x)$

Dimostrare che

1. ogni schedule ST è anche EAC (ma non necessariamente viceversa)
2. ogni schedule EAC è anche RC (ma non necessariamente viceversa)
3. il 2PL stretto permette solo schedule ST

Basi di dati II — 25 febbraio 2014

Domanda 4 (20%)
Si consideri una base di dati con le relazioni

- R1(A,B,C,D) con
  - vincolo di integrità referenziale fra l'attributo D e la chiave E della relazione R2
  - N_1=2.000.000 ennuple e fattore di blocco f_1=200
  - b=20 valori diversi sull'attributo B (tutti gli interi compresi fra 1 e b)
  - c=200.000 valori diversi sull'attributo C (tutti gli interi compresi fra 1 e c)
  - una struttura disordinata, un indice sulla chiave primaria A e un altro sull'attributo C;
- R2(E,F,G) con
  - N_2=1.000.000 ennuple e fattore di blocco f_2=10
  - una struttura disordinata, un indice sulla chiave primaria E

Supponendo che:

- gli indici abbiano tutti p=4 livelli (contando anche radice e foglie) e fattore di blocco massimo f_i=100
- il sistema esegua sempre i join come nested loop
- ogni operazione possa contare su un numero di pagine di buffer pari a circa q=150,

valutare il costo (indicandolo in modo sia simbolico sia numerico) di ciascuna delle interrogazioni seguenti:

```
select *
from R1 join R2 on D=E
where C=502
```

```
select *
from R1 join R2 on D=E
```

```
select *
from R1 join R2 on D=E
where B=5
```

Domanda 5 (20%)
Spiegare perché in un datawarehouse non è consigliabile utilizzare come identificatori le chiavi proprie delle applicazioni (numeri di matricola, codici vari o anche progressivi).

Basi di dati II — 25 febbraio 2014
Cenni sulle soluzioni (di alcune domande)

Tempo a disposizione: due ore.

Cognome _____ **Nome** _____ **Matricola** _____ **Ordin.** _____

Domanda 1 (20%)
Nella figura seguente è schematizzato un piccolissimo buffer con quattro pagine (numerate da 0 a 3), il cui stato viene descritto, per ciascuna pagina, da (i) un intero che indica il numero di pin su di essa (quindi 0 indica che la pagina è libera) (ii) un riferimento al blocco che per ultimo è stato caricato nella pagina; (iii) l'istante in cui è stato effettuato l'ultimo caricamento; (iv) l'istante in cui la pagina è stata per l'ultima volta liberata (se è libera) (v) un booleano che indica se la pagina è sporca (1 indica che è stata modificata dopo il caricamento o dopo l'ultima scrittura).

Pagina del buffer:	0	1	2	3
numero di pin sulla pagina	1	2	1	0
blocco	62	33	47	35
istante load	1	7	9	3
istante unpin				8
dirty	1	0	0	0

Si supponga ora che vengano eseguite (a partire dall'istante 10, dopo che all'istante 9 è stato caricato il blocco 47 nella pagina 2) le seguenti operazioni (in cui l'argomento delle pin e unpin è il blocco di interesse, con setXXX si indica un aggiornamento di un qualche valore nel blocco e con flushAll il flush dell'intero buffer):

Istante	10	11	12	13	14	15
Operazione	unpin(62),	pin(56),	setXXX(56,...)	unpin(56),	flushAll	setXXX(47,...)

Istante	16	17	18	19	20	21	22
Operazione	unpin(47)	pin(62),	setXXX(62,...)	unpin(62),	pin(56),	unpin(56),	pin(62)

Riempiendo la tabella seguente, indicare, per ciascuna delle strategie e per ciascuna delle pin, (i) quale pagina del buffer viene utilizzata, (ii) se il blocco corrispondente viene letto dal disco e (iii) se viene eseguita una scrittura su disco.

		naif			LRU			clock		
Istante	Operazione	Pagina	Legge?	Scrive?	Pagina	Legge?	Scrive?	Pagina	Legge?	Scrive?
11	pin(56)	0	sì	sì	3	sì	no	3	sì	no
17	pin(62)	0	sì	no	0	no	no	0	no	no
20	pin(56)	0	sì	sì	3	no	no	3	no	no
22	pin(62)	0	sì	no	0	no	no	0	no	no

Domanda 2 (20%)
Indicare il risultato di ciascuna delle seguenti interrogazioni XQuery:

```
let $i:= (3,2,1)
for $j in (1,2,3)
return <result>{$i} {$j}</result>
```

Risultato

```
<result>3 2 1 1</result>
<result>3 2 1 2</result>
<result>3 2 1 3</result>
```

```
for $i in (1,2,3)
let $j:= (3,2,1)
return <result>{$i} {$j}</result>
```

Risultato

```
<result>1 3 2 1</result>
<result>2 3 2 1</result>
<result>3 3 2 1</result>
```

```
for $i in (1,2,3)
for $j in (3,2,1)
return <result>{$i} {$j}</result>
```

Risultato

```
<result>1 3</result>
<result>1 2</result>
<result>1 3</result>
<result>2 3</result>
<result>2 2</result>
<result>2 3</result>
<result>3 3</result>
<result>3 2</result>
<result>3 3</result>
```

```
let $i:= (1,2,3)
let $j:= (3,2,1)
return <result>{$i} {$j}</result>
```

Risultato

```
<result>1 2 3 3 2 1</result>
```

Basi di dati II — 25 febbraio 2014

Domanda 3 (20%) Ai fini dello studio congiunto di affidabilità e concorrenza, è necessario considerare negli schedule anche le operazioni di commit e abort (c_i e a_i indicano rispettivamente il commit e l'abort della transazione T_i) e considerare come schedule anche i relativi prefissi (cioè porzioni di schedule eseguite fino ad un certo momento, costituite anche da transazioni non concluse). Allo scopo, si considerino le seguenti definizioni, in cui per "schedule" si intende più in generale un "prefisso di schedule" e per "coppia di transazioni T_i, T_j in s," si intende "coppia di transazioni T_i, T_j distinte (cioÃ¨ con $i \neq j$) che compaiano, almeno con un prefisso non vuoto, in s"; inoltre, si assuma che se T_i legge x da T_j, l'abort a_j non compare fra la scrittura $w_j(x)$ e la lettura $r_i(x)$:

- uno schedule s è *recoverable (RC)* se, per ogni coppia di transazioni T_i, T_j in s, vale la proprietà seguente: se T_i legge un qualche x da T_j in s e il commit c_i di T_i compare in s, allora anche il commit c_j di T_j compare in s e lo precede (in simboli $c_j <_s c_i$)
- uno schedule *evita gli abort in cascata (EAC)* se, per ogni coppia di transazioni T_i, T_j, vale la proprietà seguente: se T_i legge x da T_j in s allora $c_j <_s r_i(x)$ (il commit c_j di T_j compare nello schedule e precede la lettura di x da parte di T_i)
- uno schedule *è stretto (ST)* se, per ogni coppia di transazioni T_i, T_j, vale la proprietà seguente: se ci sono un'operazione $o_i(x)$ (lettura o scrittura) e una $w_j(x)$ con $w_j(x) <_s o_i(x)$, allora s deve includere anche c_j con $c_j <_s o_i(x)$ oppure a_j con $a_j <_s o_i(x)$

Dimostrare che

1. ogni schedule ST è anche EAC (ma non necessariamente viceversa)
2. ogni schedule EAC è anche RC (ma non necessariamente viceversa)
3. il 2PL stretto permette solo schedule ST

Basi di dati II — 25 febbraio 2014

Domanda 4 (20%)
Si consideri una base di dati con le relazioni

- R1(A,B,C,D) con
  - vincolo di integrità referenziale fra l'attributo D e la chiave E della relazione R2
  - N_1=2.000.000 ennuple e fattore di blocco f_1=200
  - b=20 valori diversi sull'attributo B (tutti gli interi compresi fra 1 e b)
  - c=200.000 valori diversi sull'attributo C (tutti gli interi compresi fra 1 e c)
  - una struttura disordinata, un indice sulla chiave primaria A e un altro sull'attributo C;
- R2(E,F,G) con
  - N_2=1.000.000 ennuple e fattore di blocco f_2=10
  - una struttura disordinata, un indice sulla chiave primaria E

Supponendo che:

- gli indici abbiano tutti p=4 livelli (contando anche radice e foglie) e fattore di blocco massimo f_i=100
- il sistema esegua sempre i join come nested loop
- ogni operazione possa contare su un numero di pagine di buffer pari a circa q=150,

valutare il costo (indicandolo in modo sia simbolico sia numerico) di ciascuna delle interrogazioni seguenti:

```
select *
from R1 join R2 on D=E
where C=502
```

$$p + \frac{N_1}{c} + \frac{N_1}{c}(p - 1 + 1) = \text{ ca. } 55$$

- accesso diretto a R1 per la selezione: p per l'indice e $\frac{N_1}{c}$ per le ennuple
- un accesso diretto a R2 per ogni ennupla nel risultato della selezione, costo unitario come sopra, con la differenza che si può ipotizzare la radice nel buffer, ma non gli altri livelli

```
select *
from R1 join R2 on D=E
```

$$\frac{N_1}{f_1} + N_1(p - 2 + 1) = 6.010.000$$

- scansione di R1: $\frac{N_1}{f_1}$
- un accesso diretto a R2 per ogni ennupla di R1: costo unitario profondità p dell'indice, meno 2 per i livelli nel buffer più uno per l'accesso al record

```
select *
from R1 join R2 on D=E
where B=5
```

$$\frac{N_1}{f_1} + \frac{N_1}{b}(p - 2 + 1) = 310.000$$

- scansione di R1 per la selezione: $\frac{N_1}{f_1}$
- un accesso diretto a R2 per ogni ennupla nel risultato della selezione, costo unitario come sopra

Domanda 5 (20%)
Spiegare perché in un datawarehouse non è consigliabile utilizzare come identificatori le chiavi proprie delle applicazioni (numeri di matricola, codici vari o anche progressivi).