Basi di dati II — Esame — 20 giugno 2014 — Compito A

Tempo a disposizione: due ore e trenta minuti (prova lunga), un'ora (prova breve).

Cognome _____ **Nome** _____ **Matricola** _____

Domanda 1 (10% per la prova lunga, 25% per la prova breve) Considerare il seguente documento XML:

```
<esami>
   <underline><corso>Matematica</corso></underline>
      <sessione data="20/02/2014" aula="N1">
         <domanda>Risolvere l'equazione x + y < 0 </domanda>
         <domanda>Dimostrare il teorema di Rolle
      </sessione>
      <sessione data="25/06/2014" aula="N16" aula="N18">
         <domanda>Dimostrare il teorema di Lagrange</domanda>
      </sessione>
</esami>
```

Indicare gli errori che fanno sì che esso non sia ben formato (ignorare l'assenza dell'intestazione `<?xml ...?>`):

Domanda 2 (10% per la prova lunga, 25% per la prova breve) Considerare il seguente DTD

```
<!ELEMENT stelledelcalcio (giocatore+)>
<!ELEMENT giocatore ( (soprann,cogn,nome?)  | soprann | (cogn,nome?)  )  >
<!ELEMENT cogn (#PCDATA) > <!ELEMENT nome (#PCDATA) > <!ELEMENT soprann (#PCDATA) >
```

che si vorrebbe usare per validare documenti come il seguente

```
<?xml version="1.0" encoding="ISO-8859-1" ?> <!DOCTYPE stelledelcalcio SYSTEM "stelledelcalcio.dtd">
<stelledelcalcio>
   <giocatore><soprann>Kaiser</soprann><cogn>Beckenbauer</cogn><nome>Franz</nome></giocatore>
   <giocatore><soprann>Cristiano Ronaldo</soprann></giocatore>
   <giocatore><cogn>Maradona</cogn></giocatore>
   <giocatore><soprann>Pelè</soprann><cogn>Arantes do Nascimento</cogn></giocatore>
   <giocatore><cogn>Cruijff</cogn><nome>Johann</nome></giocatore>
</stelledelcalcio>
```

Spiegare perché il DTD non è corretto.

Correggere il DTD (mostrare solo la riga o le righe da modificare)

Basi di dati II — 20 giugno 2014 — Compito A

Domanda 3 (10% per la prova lunga, 25% per la prova breve)
Si supponga si abbiano documenti che contengono informazioni su film, come il seguente:

```
<cineteca>
     <film>
          <titolo>Luci della città</titolo>
          <staff>
               <regista>Charlie Chaplin</regista>
               <attori>
                 <attore>Charlie Chaplin</attore>
                 <attore>Virginia Cherrill</attore>
               </attori>
          </staff>
       </film>
     <film>
          <titolo>Ninotchka</titolo>
          <staff>
               <regista>Ernst Lubitsch</regista>
               <attori>
                    <attore>Greta Garbo</attore>
                    <attore>Melvyn Douglas</attore>
               </attori>
          </staff>
       </film>
     <film>
          <titolo>Vogliamo vivere!</titolo>
          <staff>
               <regista>Ernst Lubitsch</regista>
               <regista>John Waters</regista>
               <attori>
                    <attore>Carole Lombard</attore>
                    <attore>Jack Benny</attore>
               </attori>
          </staff>
       </film>
</cineteca>
```

Per ciascuna delle seguenti interrogazioni XPath, indicare il risultato

`count(//film[staff/attori/attore="Charlie Chaplin"])`	
`count(//film[attori/attore="Charlie Chaplin"])`	
`count(//film[.//attore="Charlie Chaplin"])`	
`count(//film[//attore="Charlie Chaplin"])`	
`count(//film[.//regista])`	

Formulare in XPath l'interrogazione che mostra i titoli dei film per i quali un regista è anche uno degli attori

Basi di dati II — 20 giugno 2014 — Compito A

Domanda 4 (10% per la prova lunga, 25% per la prova breve) Considerare un documento XML con il seguente DTD

```
<!ELEMENT document (record)>
<!ELEMENT record (A,B,C,(D,E)*) >
<!ELEMENT A (#PCDATA) >
<!ELEMENT B (#PCDATA) >
<!ELEMENT C (#PCDATA) >
<!ELEMENT D (#PCDATA) >
<!ELEMENT E (#PCDATA) >
```

Si vuole memorizzare il documento in una base di dati relazionale e sono note le seguenti

- il documento contiene con $N = 400.000$ elementi `record`
- i valori dell'elemento `A`, che occupano $a = 5$ byte ciascuno, sono tutti diversi tra loro (e quindi possono essere usati per identificare gli elementi `record`)
- i valori degli elementi `B`, `C`, `D`, `E` occupano rispettivamente (supponendo per semplicità che abbiano lunghezza fissa) $b = 5$ byte, $c = 10$ byte, $d = 20$ byte, $e = 80$ byte
- ogni elementi `record` ha, in media, $k = 20$ coppie di elementi `D`, `E`
- il sistema utilizzato ha blocchi di dimensione $P = 4$ kilobyte
- si utilizzano due relazioni per la memorizzazione del documento, una con ennuple con i soli elementi `A`, `B`, `C`, e l'altra con attributi per `D` ed `E`, oltre ad attributi per l'identificazione, ad esempio `A` e un progressivo (l'equivalente della riga d'ordine o simile), lungo pure $p = 5$ byte

Indicare (in forma simbolica e numerica) il numero di accessi a memoria secondaria per ciascuna delle seguenti operazioni XPath (che verranno poi eseguite in SQL)

`//(A,B)` (restituisce gli elementi `A` e `B` di tutti gli elementi `record`)

`//` (restituisce l'intero documento; supporre che il costo aggiuntivo del join rispetto alla scansione sia trascurabile)

`//D` (restituisce tutti gli elementi `D` presenti nel documento)

`//(D,E)` (restituisce tutte le coppie di elementi `D` `E` presenti nel documento)

Basi di dati II — 20 giugno 2014 — Compito A

Domanda 5 (15%) Considerare il seguente scenario in cui tre client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si supponga che il client rilanci la stessa transazione (subito dopo l'esecuzione delle altre azioni in attesa sullo stesso dato).

client 1	client 2	client 3
		begin
		read(x)
begin		
read(x)		
	begin	
	read(x)	
x = x + 10		
write(x)		
commit		
	x = x + 20	
		read(x)
	write(x)	
	commit	
		commit

Considerare uno scheduler con controllo di concorrenza basato su **2PL** e livello di isolamento **SERIALIZABLE** su tutte le transazioni.
Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia 200. Indicare, nell'ordine, le operazioni che vengono eseguite da ciascun client, specificando, per ciascuna, il valore che viene letto o scritto. In conclusione, dire se si verificano o meno anomalie.

client 1	client 2	client 3

Indicare brevemente che cosa succede se invece la transazione sul client 3 ha livello di isolamento READ COMMITTED.

Basi di dati II — 20 giugno 2014 — Compito A

Domanda 6 (15%) Considerare il seguente scenario in cui tre client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si supponga che il client rilanci la stessa transazione (subito dopo l'esecuzione delle altre azioni in attesa sullo stesso dato).

client 1	client 2	client 3
begin		
read(x)		
	begin	
	read(x)	
x = x + 10		
write(x)		
		begin
		read(x)
	x = x + 20	
	write(x)	
	commit	
commit		
		read(x)
		commit

Considerare uno scheduler con controllo di concorrenza basato su **Multiversioni** e livello di isolamento **READ COMMITTED** su tutte le transazioni.
Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia 200. Indicare, nell'ordine, le operazioni che vengono eseguite da ciascun client, specificando, per ciascuna, il valore che viene letto o scritto. In conclusione, dire se si verificano o meno anomalie.

client 1	client 2	client 3

Basi di dati II — 20 giugno 2014 — Compito A

Domanda 7 (15%) Il *semijoin* è un'operazione simile al join, in cui del secondo operando interessano solo gli attributi di join. In concreto, se il join su $B = C$ di $R_1(AB)$ e $R_2(CD)$ è definito come

$$R_1 \text{ JOIN}_{B=C} R_2 = \{ t \text{ su } ABCD \mid \text{esistono } t_1 \in R_1 \text{ e } t_2 \in R_2 \text{ con } t[AB] = t_1,\ t[CD] = t_2 \text{ e } t_1[B] = t_2[C] \}$$

il semijoin corrispondente è definito come

$$R_1 \text{ SEMIJOIN}_{B=C} R_2 = \{ t_1 \mid t_1 \in R_1 \text{ ed esiste } t_2 \in R_2 \text{ con } t_1[B] = t_2[C] \}$$

Ad esempio:

R_1

A	B
XA	AZ
YB	AC
XB	AA
XC	AA

R_2

C	D
AA	SD
KA	FD

$R_1 \text{ SEMIJOIN}_{B=C} R_2$

A	B
XB	AA
XC	AA

Considerare le relazioni R1 ed R2 schematizzate sotto. I riquadri interni indicano i blocchi e il numero a fianco a ciascun riquadro indica l'indirizzo del blocco. Quindi R1 occupa $B_1 = 8$ blocchi e R2 ne occupa $B_2 = 8$.

Relazione R1

20	XA	AZ	21	XB	AA	22	XC	AA	23	XD	AD	24	XE	AA	25	XF	BZ	26	XG	BA	27	XH	BB
	YB	AC		YC	DZ		YD	BB		YE	BZ		YF	AZ		YG	BY		YH	DZ		YI	DZ

Relazione R2

40	BB	RF	41	AC	VV	42	BA	GV	43	KD	TT	44	KF	DF	45	JB	RF	46	AA	SD	47	AD	RX
	KA	FD		KB	JH		KC	HG		KE	SD		JA	IU		JL	VC		LA	LK		LB	MN

Si supponga di disporre di un buffer di $p = 5$ pagine. Supporre anche che gli attributi abbiano tutti la stessa lunghezza e che lo spazio per le "informazioni di servizio" sia trascurabile. Quindi, se un blocco (e quindi una pagina di buffer) contiene due record, una pagina di buffer può contenere quattro valori.
Considerare l'esecuzione del semijoin di R1 ed R2, sulla base dei valori del secondo attributo di R1 e del primo di R2, con il metodo nested loop senza utilizzo di indici. Supporre che non serva memorizzare il risultato e che quindi esso possa essere prodotto una ennupla alla volta (approccio "pipelining"). Tenere conto del fatto che il secondo attributo della seconda relazione non interessa per il risultato.

Indicare, nell'ordine, le prime quattro ennuple che vengono prodotte

Mostrare il contenuto del buffer al momento in cui viene prodotta la prima ennupla del risultato.

Indicare gli indirizzi dei blocchi effettivamente letti da memoria secondaria e caricati nel buffer (nell'ordine) per produrre le prime quattro ennuple del risultato.

Indicare il numero complessivo di accessi a memoria secondaria necessari per eseguire questo semijoin (indicare formula e numero)

Basi di dati II — 20 giugno 2014 — Compito A

Domanda 8 (15%) Considerare un sistema distribuito su cui vengono eseguite due transazioni che coinvolgono tre nodi, un coordinatore C (lo stesso per entrambe le transazioni) e due partecipanti P1 e P2. Dopo la richiesta del coordinatore C di `prepare` (abbreviata con `prep`) per la prima transazione, i due partecipanti ricevono e rispondono correttamente, e uno dei due, P1, va in crash subito dopo aver risposto. Il coordinatore riceve le risposte, prende la decisione, invia i relativi messaggi (questi passi **non** sono stati indicati sotto e vanno quindi scritti) e subito dopo va anch'esso in crash (quindi senza fare in tempo a ricevere le conferme). Viceversa, per la seconda transazione, il coordinatore invia il messaggio di `prepare` ma non fa in tempo a ricevere le risposte. Indicare, nello schema sottostante, una possibile sequenza di scritture sui log e invio di messaggi (che includa anche i passi sopra illustrati), supponendo che entrambi i nodi siano ripristinati abbastanza presto (ma che il coordinatore perda alcuni messaggi di risposta inviati ad esso a seguito del commit). Per i messaggi si usi la notazione *tipo(transaz)*→*destinatari* (come nell'esempio: `prepare(T1)`→`P1,P2`). Supporre che nel log del coordinatore si scrivano solo i record di `prepare`, `commit` e `complete`, mentre i messaggi sono gestiti in memoria. Indicare ragionevoli istanti per i timeout, che permettano di concludere entrambe le transazioni.

Nodo C		Nodo P1		Nodo P2	
Log	Messaggi	Log	Messaggi	Log	Messaggi
`prep(T1,P1,P2)`					
	`prep(T1)`→`P1,P2`				
			crash		
`prep(T2,P1,P2)`					
	`prep(T2)`→`P1,P2`				
crash					
restart					
			restart		

Basi di dati II — Esame — 20 giugno 2014 — Compito A

Cenni sulle soluzioni (solo Compito A, le varianti del testo sono in rosso)

Tempo a disposizione: due ore e trenta minuti (prova lunga), un'ora (prova breve).

Cognome _____ **Nome** _____ **Matricola** _____

Domanda 1 (10% per la prova lunga, 25% per la prova breve) Considerare il seguente documento XML:

```
<esami>
   <underline><corso>Matematica</corso></underline>
      <sessione data="20/02/2014" aula="N1">
         <domanda>Risolvere l'equazione x + y < 0 </domanda>
         <domanda>Dimostrare il teorema di Rolle
      </sessione>
      <sessione data="25/06/2014" aula="N16" aula="N18">
         <domanda>Dimostrare il teorema di Lagrange</domanda>
      </sessione>
</esami>
```

Indicare gli errori che fanno sì che esso non sia ben formato (ignorare l'assenza dell'intestazione `<?xml ...?>`):

- un elemento non può avere due attributi con lo stesso nome (`aula`)
- ad ogni tag di apertura deve corrispondere un tag di chiusura (e ciò non accade per `domanda`)
- il simbolo `<` non può comparire nel testo di un elemento; si deve usare la entity reference `<`

Domanda 2 (10% per la prova lunga, 25% per la prova breve) Considerare il seguente DTD

```
<!ELEMENT stelledelcalcio (giocatore+)>
<!ELEMENT giocatore ( (soprann,cogn,nome?)  | soprann | (cogn,nome?)  )  >
<!ELEMENT cogn (#PCDATA) > <!ELEMENT nome (#PCDATA) > <!ELEMENT soprann (#PCDATA) >
```

che si vorrebbe usare per validare documenti come il seguente

```
<?xml version="1.0" encoding="ISO-8859-1" ?> <!DOCTYPE stelledelcalcio SYSTEM "stelledelcalcio.dtd">
<stelledelcalcio>
   <giocatore><soprann>Kaiser</soprann><cogn>Beckenbauer</cogn><nome>Franz</nome></giocatore>
   <giocatore><soprann>Cristiano Ronaldo</soprann></giocatore>
   <giocatore><cogn>Maradona</cogn></giocatore>
   <giocatore><soprann>Pelè</soprann><cogn>Arantes do Nascimento</cogn></giocatore>
   <giocatore><cogn>Cruijff</cogn><nome>Johann</nome></giocatore>
</stelledelcalcio>
```

Spiegare perché il DTD non è corretto.
Possibile risposta
La specifica è ambigua in quanto l'espressione regolare è non deterministica: leggendo un documento, il parser, trovato un elemento `soprann` non saprebbe se proseguire con la prima sottoespressione o la seconda

Correggere il DTD (mostrare solo la riga o le righe da modificare)
Risposta, la seconda riga del DTD

```
<!ELEMENT giocatore ( (soprann,(cogn,nome?)?)  | (cogn,nome?)  )  >
```

Nota bene, le seguenti risposte sono sbagliate

```
<!ELEMENT giocatore ( (soprann,cogn?,nome?)  | (cogn,nome?)  )  >
<!ELEMENT giocatore ( (soprann?,(cogn,nome?)?)  )  >
```

perché permettono anche casi non permessi dall'espressione originaria (il primo `soprann,nome` e il secondo l'elemento vuoto

Basi di dati II — 20 giugno 2014 — Compito A

Domanda 3 (10% per la prova lunga, 25% per la prova breve)
Si supponga si abbiano documenti che contengono informazioni su film, come il seguente:

```
<cineteca>
     <film>
          <titolo>Luci della città</titolo>
          <staff>
               <regista>Charlie Chaplin</regista>
               <attori>
                 <attore>Charlie Chaplin</attore>
                 <attore>Virginia Cherrill</attore>
               </attori>
          </staff>
       </film>
     <film>
          <titolo>Ninotchka</titolo>
          <staff>
               <regista>Ernst Lubitsch</regista>
               <attori>
                    <attore>Greta Garbo</attore>
                    <attore>Melvyn Douglas</attore>
               </attori>
          </staff>
        </film>
     <film>
          <titolo>Vogliamo vivere!</titolo>
          <staff>
               <regista>Ernst Lubitsch</regista>
               <regista>John Waters</regista>
               <attori>
                    <attore>Carole Lombard</attore>
                    <attore>Jack Benny</attore>
               </attori>
          </staff>
        </film>
</cineteca>
```

Per ciascuna delle seguenti interrogazioni XPath, indicare il risultato

`count(//film[staff/attori/attore="Charlie Chaplin"])`	1
`count(//film[attori/attore="Charlie Chaplin"])`	0
`count(//film[.//attore="Charlie Chaplin"])`	1
`count(//film[//attore="Charlie Chaplin"])`	3
`count(//film[.//regista])`	3

Formulare in XPath l'interrogazione che mostra i titoli dei film per i quali un regista è anche uno degli attori

`//film[.//regista=.//attore]/titolo` oppure
`//film[staff/regista=staff/attori/attore]/titolo` oppure
`//film[./staff/regista=./staff/attori/attore]/titolo`

Basi di dati II — 20 giugno 2014 — Compito A

Domanda 4 (10% per la prova lunga, 25% per la prova breve) Considerare un documento XML con il seguente DTD

```
<!ELEMENT document (record)>
<!ELEMENT record (A,B,C,(D,E)*) >
<!ELEMENT A (#PCDATA) >
<!ELEMENT B (#PCDATA) >
<!ELEMENT C (#PCDATA) >
<!ELEMENT D (#PCDATA) >
<!ELEMENT E (#PCDATA) >
```

Si vuole memorizzare il documento in una base di dati relazionale e sono note le seguenti

- il documento contiene con $N = 400.000$ elementi `record`
- i valori dell'elemento `A`, che occupano $a = 5$ byte ciascuno, sono tutti diversi tra loro (e quindi possono essere usati per identificare gli elementi `record`)
- i valori degli elementi `B`, `C`, `D`, `E` occupano rispettivamente (supponendo per semplicità che abbiano lunghezza fissa) $b = 5$ byte, $c = 10$ byte, $d = 20$ byte, $e = 80$ byte
- ogni elementi `record` ha, in media, $k = 20$ coppie di elementi `D`, `E`
- il sistema utilizzato ha blocchi di dimensione $P = 4$ kilobyte
- si utilizzano due relazioni per la memorizzazione del documento, una con ennuple con i soli elementi `A`, `B`, `C`, e l'altra con attributi per `D` ed `E`, oltre ad attributi per l'identificazione, ad esempio `A` e un progressivo (l'equivalente della riga d'ordine o simile), lungo pure $p = 5$ byte

Indicare (in forma simbolica e numerica) il numero di accessi a memoria secondaria per ciascuna delle seguenti operazioni XPath (che verranno poi eseguite in SQL)

`//(A,B)` (restituisce gli elementi `A` e `B` di tutti gli elementi `record`)

L'operazione deve visitare tutti i blocchi della prima relazione, che sono:

$N \times (a + b + c)/P = 420.000 \times 20/4000 = 2000$

`//` (restituisce l'intero documento; supporre che il costo aggiuntivo del join rispetto alla scansione sia trascurabile)

L'operazione deve visitare tutti i blocchi di entrambe le relazioni. Quelli della seconda sono:

$N \times k \times (a + p + d + e)/P = 40.000 \times 2 \times (5 + 5 + 20 + 80)/4000 =$ ca 220.000

`//D` (restituisce tutti gli elementi `D` presenti nel documento)

L'operazione deve visitare tutti i blocchi della seconda relazione — vedi sopra

`//(D,E)` (restituisce tutte le coppie di elementi `D` `E` presenti nel documento)

L'operazione deve visitare tutti i blocchi della seconda relazione — vedi sopra

Basi di dati II — 20 giugno 2014 — Compito A

Domanda 5 (15%) Considerare il seguente scenario in cui tre client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si supponga che il client rilanci la stessa transazione (subito dopo l'esecuzione delle altre azioni in attesa sullo stesso dato).

client 1	client 2	client 3
		begin
		read(x)
begin		
read(x)		
	begin	
	read(x)	
x = x + 10		
write(x)		
commit		
	x = x + 20	
		read(x)
	write(x)	
	commit	
		commit

Considerare uno scheduler con controllo di concorrenza basato su **2PL** e livello di isolamento **SERIALIZABLE** su tutte le transazioni.
Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia 200. Indicare, nell'ordine, le operazioni che vengono eseguite da ciascun client, specificando, per ciascuna, il valore che viene letto o scritto. In conclusione, dire se si verificano o meno anomalie.

client 1	client 2	client 3
		begin
		read(x) *legge 200*
begin		
read(x) *legge 200*		
	begin	
	read(x) *legge 200*	
x = x + 10		
xlock(x) *attesa*		
	x = x + 20	
		read(x) *legge 200*
	xlock(x) *attesa*	
		commit
abort		
	write(x) *scrive 220*	
	commit	
read(x) *legge 220*		
x = x + 10		
write(x) *scrive 230*		
commit		

Indicare brevemente che cosa succede se invece la transazione sul client 3 ha livello di isolamento READ COMMITTED.
Risposta
Il cliente 3 rilascia il lock dopo la prima lettura e permette quindi alle altre due transazioni di procedere (e almeno una di esse ci riesce, dopo lo stallo). La seconda lettura legge poi un valore diverso

Basi di dati II — 20 giugno 2014 — Compito A

Domanda 6 (15%) Considerare il seguente scenario in cui tre client diversi inviano richieste ad un gestore del controllo di concorrenza. Ciascun client può inviare una richiesta solo dopo che è stata eseguita o rifiutata la precedente (se invece una richiesta viene bloccata da un lock, allora il client rimane inattivo fino alla concessione o allo scadere del timeout). Si supponga che, in caso di stallo, abortisca la transazione che ha avanzato la richiesta per prima. In caso di abort, si supponga che il client rilanci la stessa transazione (subito dopo l'esecuzione delle altre azioni in attesa sullo stesso dato).

client 1	client 2	client 3
begin		
read(x)		
	begin	
	read(x)	
x = x + 10		
write(x)		
		begin
		read(x)
	x = x + 20	
	write(x)	
	commit	
commit		
		read(x)
		commit

Considerare uno scheduler con controllo di concorrenza basato su **Multiversioni** e livello di isolamento **READ COMMITTED** su tutte le transazioni.
Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia 200. Indicare, nell'ordine, le operazioni che vengono eseguite da ciascun client, specificando, per ciascuna, il valore che viene letto o scritto. In conclusione, dire se si verificano o meno anomalie.

client 1	client 2	client 3
begin		
read(x) *legge 200*		
	begin	
	read(x) *legge 200*	
x = x + 10		
write(x) *scrive 210*		
		begin
		read(x) *legge 200*
	x = x + 20	
	xlock(x) *attesa*	
commit		
	write(x) *scrive 220*	
	commit	
		read(x) *legge 220*
		commit

Basi di dati II — 20 giugno 2014 — Compito A

Domanda 7 (15%) Il *semijoin* è un'operazione simile al join, in cui del secondo operando interessano solo gli attributi di join. In concreto, se il join su $B = C$ di $R_1(AB)$ e $R_2(CD)$ è definito come

$$R_1 \text{ JOIN}_{B=C} R_2 = \{ t \text{ su } ABCD \mid \text{esistono } t_1 \in R_1 \text{ e } t_2 \in R_2 \text{ con } t[AB] = t_1,\ t[CD] = t_2 \text{ e } t_1[B] = t_2[C] \}$$

il semijoin corrispondente è definito come

$$R_1 \text{ SEMIJOIN}_{B=C} R_2 = \{ t_1 \mid t_1 \in R_1 \text{ ed esiste } t_2 \in R_2 \text{ con } t_1[B] = t_2[C] \}$$

Ad esempio:

R_1

A	B
XA	AZ
YB	AC
XB	AA
XC	AA

R_2

C	D
AA	SD
KA	FD

$R_1 \text{ SEMIJOIN}_{B=C} R_2$

A	B
XB	AA
XC	AA

Considerare le relazioni R1 ed R2 schematizzate sotto. I riquadri interni indicano i blocchi e il numero a fianco a ciascun riquadro indica l'indirizzo del blocco. Quindi R1 occupa $B_1 = 8$ blocchi e R2 ne occupa $B_2 = 8$.

Relazione R1

20	21	22	23	24	25	26	27
XA AZ YB AC	XB AA YC DZ	XC AA YD BB	XD AD YE BZ	XE AA YF AZ	XF BZ YG BY	XG BA YH DZ	XH BB YI DZ

Relazione R2

40	41	42	43	44	45	46	47
BB RF KA FD	AC VV KB JH	BA GV KC HG	KD TT KE SD	KF DF JA IU	JB RF JL VC	AA SD LA LK	AD RX LB MN

Si supponga di disporre di un buffer di $p = 5$ pagine. Supporre anche che gli attributi abbiano tutti la stessa lunghezza e che lo spazio per le "informazioni di servizio" sia trascurabile. Quindi, se un blocco (e quindi una pagina di buffer) contiene due record, una pagina di buffer può contenere quattro valori.
Considerare l'esecuzione del semijoin di R1 ed R2, sulla base dei valori del secondo attributo di R1 e del primo di R2, con il metodo nested loop senza utilizzo di indici. Supporre che non serva memorizzare il risultato e che quindi esso possa essere prodotto una ennupla alla volta (approccio "pipelining"). Tenere conto del fatto che il secondo attributo della seconda relazione non interessa per il risultato.

Indicare, nell'ordine, le prime quattro ennuple che vengono prodotte

(YB,AC) (XB,AA) (XC,AA) (YD,BB)

Mostrare il contenuto del buffer al momento in cui viene prodotta la prima ennupla del risultato.

BB KA AC KB	BA KC KD KE	KF JA JB JL	AA LA AD LB	XA AZ YB AC

Indicare gli indirizzi dei blocchi effettivamente letti da memoria secondaria e caricati nel buffer (nell'ordine) per produrre le prime quattro ennuple del risultato.

40, 41, 42, 43, 44, 45, 46, 47, 20, 21, 22

Indicare il numero complessivo di accessi a memoria secondaria necessari per eseguire questo semijoin (indicare formula e numero)

$B_1 + B_2 = 16$

Basi di dati II — 20 giugno 2014 — Compito A

Domanda 8 (15%) Considerare un sistema distribuito su cui vengono eseguite due transazioni che coinvolgono tre nodi, un coordinatore C (lo stesso per entrambe le transazioni) e due partecipanti P1 e P2. Dopo la richiesta del coordinatore C di `prepare` (abbreviata con `prep`) per la prima transazione, i due partecipanti ricevono e rispondono correttamente, e uno dei due, P1, va in crash subito dopo aver risposto. Il coordinatore riceve le risposte, prende la decisione, invia i relativi messaggi (questi passi **non** sono stati indicati sotto e vanno quindi scritti) e subito dopo va anch'esso in crash (quindi senza fare in tempo a ricevere le conferme). Viceversa, per la seconda transazione, il coordinatore invia il messaggio di `prepare` ma non fa in tempo a ricevere le risposte. Indicare, nello schema sottostante, una possibile sequenza di scritture sui log e invio di messaggi (che includa anche i passi sopra illustrati), supponendo che entrambi i nodi siano ripristinati abbastanza presto (ma che il coordinatore perda alcuni messaggi di risposta inviati ad esso a seguito del commit). Per i messaggi si usi la notazione *tipo(transaz)*→*destinatari* (come nell'esempio: `prepare(T1)`→`P1,P2`). Supporre che nel log del coordinatore si scrivano solo i record di `prepare`, `commit` e `complete`, mentre i messaggi sono gestiti in memoria. Indicare ragionevoli istanti per i timeout, che permettano di concludere entrambe le transazioni.

Nodo C		Nodo P1		Nodo P2	
Log	Messaggi	Log	Messaggi	Log	Messaggi
`prep(T1,P1,P2)`					
	`prep(T1)`→`P1,P2`				
		`ready(T1)`		`ready(T1)`	
			`ready(T1)`→`C`		`ready(T1)`→`C`
		crash			
`commit(T1)`					
	`commit(T1)`→`P1,P2`				
				`commit(T1)`	
					`ack(T1)`→`C`
`prep(T2,P1,P2)`					
	`prep(T2)`→`P1,P2`				
				`ready(T2)`	
					`ready(T2)`→`C`
	crash				
	restart				
	`commit(T1)`→`P1,P2`				
`abort(T2)`					
	`abort(T2)`→`P1,P2`				
					`ack(T1)`→`C`
				`abort(T2)`	
					`ack(T2)`→`C`
		restart			
	`commit(T1)`→`P1`				
	`abort(T2)`→`P1`				
		`commit(T1)`			
			`ack(T1)`→`C`		
		`abort(T2)`			
			`ack(T2)`→`C`		
`complete(T1)`					
`complete(T2)`					