

Basi di dati II
Esame — 26 febbraio 2013
Cenni sulle soluzioni

Rispondere su questo fascicolo. Tempo a disposizione: due ore e trenta minuti.

Cognome _____ Nome _____ Matricola _____

Domanda 1 (15%)

Si consideri un DBMS che preveda, in aggiunta alle tradizionali operazioni di lettura e scrittura, anche un'operazione *increment*(x, i) che modifica il valore di x (intero) incrementandolo di i , senza renderlo in disponibile alla transazione (il che richiederebbe un'esplicita operazione di lettura). Per includere questo tipo di operazioni, in aggiunta a quelle di lettura e scrittura, è stato proposto di utilizzare un terzo tipo di lock, detto i-lock, con la seguente regola di compatibilità (in aggiunta a quelle note per r-lock e w-lock): due i-lock su un oggetto sono fra loro compatibili, mentre un i-lock è incompatibile sia con un r-lock sia con un w-lock. Mostrare che:

1. schedule 2-PL che soddisfino le regole di compatibilità note estese con quelle sopra citate risultano serializzabili;
2. questa tecnica può generare transazioni a maggiore livello di concorrenza (cioè con un throughput maggiore).

1.

Facendo riferimento alla view serializzabilità, cambiare l'ordine di operazioni di incremento non modifica né i risultati che vengono forniti all'esterno (perché gli incrementi non acquisiscono valore quindi non permettono di comunicare niente) né le interazioni fra le transazioni e quindi le modifiche sulla base di dati.

Con riferimento alla conflict-serializzabilità, di conseguenza, due incrementi non sono in conflitto fra loro.

Le regole sull'i-lock permettono appunto di non forzare l'ordine sugli incrementi, ma fra incrementi e altre operazioni

2.

Con questa tecnica, operazioni di incremento multiple non si rallentano a vicenda, mentre con un approccio tradizionale ognuna di esse sarebbe costituita da una lettura e una scrittura, con conseguenti rallentamenti dovuti ai lock esclusivi.

Domanda 2 (10%)

Considerare un'operazione di prelevamento di contante presso uno sportello Bancomat (che operi come terminale con capacità elaborativa ridotta, ma non nulla). Essa prevede, fra le altre, le due azioni seguenti:

- emissione delle banconote.
- commit della transazione;

Rispondere alle seguenti domande:

1. quale delle due azioni deve essere eseguita per prima?
2. spiegare come potrebbe essere gestito, in tal caso, un guasto (caduta del sistema centrale, del terminale o del collegamento) verificatosi fra la prima e la seconda azione
3. spiegare perché se le azioni venissero eseguite in ordine inverso (rispetto a quello indicato nella prima risposta) si avrebbero conseguenze inaccettabili?

1.
prima il commit e poi l'emissione

2.
La transazione verrebbe registrata (con il commit). L'esecuzione dell'operazione da parte del componente meccanico invece no e quindi potrebbe venire richiesta, al riavvio del sistema, una "transazione compensativa" che porti al riaccredito delle somme non erogate.

3.
Perché un crash dopo l'erogazione non permetterebbe di tenere traccia dell'erogazione né, tantomeno, di annullarla.

Domanda 3 (15%)

Considerare un sistema distribuito su cui viene eseguita una transazione che coinvolge tre nodi, un coordinatore N1 e due partecipanti N2 e N3. Dopo la richiesta di **prepare** da parte del coordinatore, il partecipante N3 va in crash senza ricevere il messaggio, mentre il partecipante N2 fa in tempo a ricevere il messaggio e a rispondere positivamente ma va in crash poco dopo, non ricevendo la prima notifica della decisione. Indicare, nello schema sottostante, una possibile sequenza di scritture sui log e invio di messaggi, supponendo che entrambi i nodi siano ripristinati abbastanza presto. Per semplicità, si fa riferimento ad una sola transazione e quindi non c'è bisogno di indicarla. Per i messaggi si usi la notazione *tipo*→*destinatari* (come nell'esempio: **prepare**→N2,N3). Supporre che timeout per le varie fasi scattino all'incirca negli istanti indicati a sinistra della tabella.

Nodo N1		Nodo N2		Nodo N3	
Log	Messaggi	Log	Messaggi	Log	Messaggi
<div> <div>prepare(N2,N3)</div> <div>abort</div> <div>complete</div> </div>	prepare→N2,N3	ready	ready→N1	crash	
	abort→N2,N3		crash		
	abort→N2,N3	abort	ack→N1		
t ₂			restart		
t ₃	abort→N3			restart	
				abort	ack→N1

Eventuali commenti:

N1 deve ricevere un **ack** anche da N3, prima di concludere con **complete**, perché N3 potrebbe essere in stato di **ready**

Domanda 4 (20%)

Si considerino un sistema con blocchi di dimensione $B = 1000$ byte e puntatori ai blocchi di $Q = 2$ byte e una relazione $T(\underline{A}, C, D)$ di cardinalità pari circa a $R = 2.000.000$, con ennuple di $L = 20$ byte e campo chiave A di $L_A = 5$ byte e campo C di $L_C = 8$ byte. Il campo C non è chiave e ogni suo valore è presente, mediamente, in $e = 6$ ennuple. Valutare i pro e i contro relativamente alla presenza di un indice secondario sulla chiave A e di un altro, pure secondario, su C , in presenza del seguente carico applicativo:

1. inserimento di una nuova ennupla (con verifica del soddisfacimento del vincolo di chiave), con frequenza $f_1 = 2.000$
2. ricerca di una ennupla sulla base del valore della chiave A , con frequenza $f_2 = 1.000$
3. ricerca di ennuple sulla base del valore di C , con frequenza $f_3 = 10$

Ragionare in termini di costo degli accessi a memoria secondaria, assumendo disponibilità di buffer che permettano di mantenere stabilmente in memoria due livelli per ciascun indice e considerando che la relazione possa essere memorizzata in forma contigua (assumendo che il tempo di posizionamento della testina sia $r = 100$ volte maggiore del tempo di lettura). Trascurare le problematiche relative alla concorrenza e considerare il costo della lettura pari a quello della scrittura. Rispondere negli spazi sottostanti, in forma sia simbolica sia numerica.

Costo scansione sequenziale (SEQ) in multipli del tempo di posizionamento

numero di blocchi: $NB = \frac{R}{B/L} = 40.000$

numero di accessi: $1 + (NB - 1) \times \frac{1}{r} \approx 400$

Numero livelli indice su A (PA)

il fattore di blocco dell'indice è $1.000/7$, circa 160 e quindi, con un riempimento di circa il 60-70%, il fan-out è circa 100 e quindi i livelli necessari sono 4

Numero livelli indice su C (PC)

il fattore di blocco dell'indice è $1.000/10$, circa 100 e quindi, con un riempimento di circa il 60-70%, il fan-out è circa 65 e quindi i livelli necessari sono pure 4

	nessun indice	indice solo su A	indice solo su C	indice su A e su C
Op. 1 Costo unit.	SEQ = 400	PA - 2 + 1 + 1 = 4	SEQ + ... = ~ 400	PA - 2 + PC - 2 + 1 + 1 = 6
Op. 2 Costo unit.	SEQ = 400	PA - 2 + 1 = 3	SEQ = 400	PA - 2 + 1 = 3
Op. 3 Costo unit.	SEQ = 400	SEQ = 400	PC - 2 + e = 8	PC - 2 + e = 8
Costo complessivo	ca 1.200.000	ca 19.000	ca 1.200.000	ca 15.000

Domanda 5 (15%)

Considerare la relazione sotto schematizzata, definita su vari attributi, uno dei quali è la chiave, i cui valori sono mostrati. Supponendo una disponibilità di buffer abbastanza ampia, ma non sufficiente a caricare in memoria l'intera relazione (supporre ad esempio una disponibilità di 8 buffer, con un fattore di blocco pari a 2 e quindi uno spazio occupato dalla relazione pari a 16 blocchi), considerare l'esecuzione di un mergesort a più vie (e due passate) sulla relazione e mostrare lo stato delle strutture in memoria centrale e secondaria dopo l'esecuzione di sei chiamate al metodo `next()` sullo scan che implementa il mergesort. In particolare, mostrare i “run” (cioè le porzioni di file ordinate durante prima passata) memorizzati su disco e i buffer in memoria centrale, evidenziando per ciascun buffer il record corrente. Mostrare anche i record prodotti dalle prime sei chiamate di `next()`.

Run su disco

321	...
411	...
111	...
621	...
811	...
221	...
521	...
711	...
452	...
182	...
662	...
712	...
812	...
532	...
122	...
162	...
583	...
633	...
123	...
243	...
343	...
473	...
723	...
873	...
234	...
124	...
534	...
664	...
354	...
464	...
724	...
854	...

(totale
32 ennuple,
16 blocchi)

Buffer

111	...
211	...

←

182	...
452	...

←

123	...
243	...

←

124	...
234	...

←

Record prodotti dalle
prime 6 `next()`

111	...
122	...
123	...
124	...
162	...
182	...

Il numero ideale di buffer (da utilizzare sia nella prima sia nella seconda passata) è pari alla radice quadrata del numero dei blocchi e quindi a quattro.

Nella prima passata, si costruiscono quindi quattro run ordinati, ciascuno a partire da quattro blocchi del file originario. L'ordinamento di ciascun run può essere effettuato in memoria centrale, usando quattro buffer. Poiché il risultato della prima passata viene materializzato, vengono mostrati i run ordinati.

Nella seconda passata, si carica un blocco per ciascuno dei run e si fa il merge usando nuovamente quattro buffer. La seconda passata viene svolta in pipeline, e quindi i buffer sono mostrati con il contenuto che hanno quando è stato appena prodotto il sesto record.

Domanda 6 (10%)

Considerare il documento XML qui sotto e definire uno schema XSD per il quale esso sia valido.

```
<?xml version="1.0" encoding="UTF-8"?>
<students>
  <student>
    <firstName> Paolo </firstName>
    <lastName> Neri </lastName>
    <studentNumber> 281283 </studentNumber>
    <birth>
      <date> 13/06/11 </date>
      <city> Roma </city>
    </birth>
    <courses>
      <course>
        <name> Programmazione Orientata agli Oggetti </name>
        <shortName> POO </shortName>
        <record>
          <grade> 28 </grade>
          <date> 13/06/11 </date>
        </record>
      </course>
      <course>
        <name> Analisi e progettazione del software </name>
        <shortName> APS </shortName>
      </course>
      ...
    </courses>
  </student>
  <student>
    <firstName> Luca </firstName>
    <lastName> Verdi </lastName>
    <studentNumber> 281111 </studentNumber>
    <courses>
    </courses>
  </student>
  <student>
    ...
  </student>
</students>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="students">
    <xsd:complexType>
      <xsd:sequence minOccurs="0" maxOccurs="unbounded">
        <xsd:element ref="student" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="student">
    <xsd:complexType>
```

```

        <xsd:sequence>
            <xsd:element name="firstName" type="xsd:string" />
            <xsd:element name="lastName" type="xsd:string" />
            <xsd:element name="studentNumber" type="xsd:string" />
            <xsd:element ref="birth" minOccurs="0"/>
            <xsd:element ref="courses" />
        </xsd:sequence>
    </xsd:complexType>
    <xsd:unique name="studentID">
        <xsd:selector xpath="." />
        <xsd:field xpath="id" />
    </xsd:unique>
</xsd:element>
<xsd:element name="record">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="grade" type="xsd:string" />
            <xsd:element name="date" type="xsd:string" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="course">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="name" type="xsd:string" />
            <xsd:element name="shortName" type="xsd:string" />
            <xsd:element ref="record" minOccurs="0" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="courses">
    <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
            <xsd:element ref="course" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
<xsd:element name="birth">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="date" type="xsd:string" />
            <xsd:element name="city" type="xsd:string" />
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
</xsd:schema>

```

Domanda 7 (15%)

Con riferimento a documenti come quello mostrato nella Domanda 2 (supponendolo memorizzato nel file `esame.xml`), rispondere alle seguenti interrogazioni

1. In XPath, trovare gli studenti che hanno superato APS con voto superiore a 27

```
//student[.//course[shortName="APS" and .//grade>27]]
```

2. In XPath, trovare gli studenti che hanno superato almeno tre esami

```
//student[(.//record)[3]]
```

3. In XQuery, trovare gli studenti che hanno superato due esami con uno stesso voto

```
xquery version "1.0";  
for $s in fn:doc("esame.xml")//student  
let $e1 := $s//course  
let $e2 := $s//course  
  
where $e2 >> $e1 and $e1//grade = $e2//grade  
  
return $s
```