Basi di dati II — 21 febbraio 2017

Tempo a disposizione: un'ora e quarantacinque minuti.

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

Domanda 1 (15%)
Considerare un sistema con dischi con $N = 1000$ blocchi per traccia

- tempo medio di posizionamento della testina (tempo di seek) $t_S = 6$ msec
- tempo medio di latenza (attesa dovuta alla rotazione) $t_L = 2$ msec
- tempo minimo di lettura di un blocco $t_B = 10$ μsec

Rispondere alle seguenti domande mostrando formula e valore numerico

1. Qual è il tempo medio necessario per leggere un blocco del quale sia dato l'indirizzo fisico?

2. Qual è il tempo medio necessario per la scansione sequenziale di un file costituito da $F = 200$ blocchi contigui, non letti di recente?

3. Qual è il tempo che si può ipotizzare necessario per eseguire $m = 20.000$ accessi diretti in tempi ravvicinati a record di un file (molto grande) attraverso un indice che abbia profondità $p = 4$, fan-out $f_I = 50$, con disponibilità di circa $P = 4000$ pagine di buffer?

4. Qual è il tempo che si può ipotizzare necessario per eseguire tre accessi diretti a record di un file attraverso un indice che abbia profondità $p = 4$ e fan-out (fattore di blocco dell'indice) $f_I = 50$, non usato di recente?

Basi di dati II — 21 febbraio 2017

Domanda 2 (15%)
Si consideri un DBMS che preveda, in aggiunta alle tradizionali operazioni di lettura e scrittura, anche un'operazione INCREMENTA*(x,i)* che modifica il valore di x (intero) incrementandolo di i, senza renderlo 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 inc-lock, con la seguente regola di compatibilità (in aggiunta a quelle note per r-lock e w-lock): due inc-lock su un oggetto sono fra loro compatibili, mentre un inc-lock è incompatibile sia con un r-lock sia con un w-lock.

1. Spiegare intuitivamente perché questa tecnica può generare transazioni a maggiore livello di concorrenza (cioè con un throughput maggiore).

2. Dimostrare (almeno intuitivamente) che schedule 2-PL che soddisfino le regole di compatibilità note estese con quelle sopra citate risultano view-serializzabili.

Basi di dati II — 21 febbraio 2017

Domanda 3 (25%) Sia data una relazione $R(\underline{A}, B, C)$ contenente circa $N = 10.000.000$ ennuple di $r = 20$ byte ciascuna, di cui $k = 4$ per la chiave A, che contiene valori interi quasi consecutivi, da 1 a poco più di 10.000.000. Supporre che i blocchi abbiano dimensione $B = 2$KB, approssimabile come 2.000, che i puntatori ai record abbiano lunghezza $p = 6$; e che i nodi intermedi degli indici possano essere contenuti nei buffer.
Indicare il costo prevedibile per le seguenti operazioni

1. `SELECT * FROM R WHERE A >= 1000 AND A <=2000`
2. `SELECT COUNT(*) FROM R WHERE A >= 1000 AND A <=2000`
3. `SELECT * FROM R WHERE A = 1500`

in ciascuno dei seguenti casi:

(a) indice primario (sparso) su A realizzato con B+-tree;
(b) indice secondario (denso) su A realizzato con B+-tree.

Riportare le risposte nella tabella sottostante, indicando formula e valore numerico (con brevissimo commento, se necessario)

	Risposte
caso (a) op.1	
caso (b) op.1	
caso (a) op.2	
caso (b) op.2	
caso (a) op.3	
caso (b) op.3	

Basi di dati II — 21 febbraio 2017

Domanda 4 (25%)
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 + 20 write(x) commit	
x = x + 10 write(x) commit		
		read(x) commit

Considerare uno scheduler con controllo di concorrenza basato su **Multversioni** (come in Postgres) e livello di isolamento **SERIALIZABLE** sulle prime due transazioni e **READ COMMITTED** sulla terza. Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia **1000**. 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

Si verificano anomalie?

Basi di dati II — 21 febbraio 2017

Considerare nuovamente lo scenario della pagina precedente, ripetuto qui sotto per comodità.

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

Considerare uno scheduler con controllo di concorrenza ancora basato su **Multversioni** (come in Postgres) ma con livello di isolamento **READ COMMITTED** sulle prime due transazioni e **SERIALIZABLE** sulla terza. Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia ancora **1000**.

client 1	client 2	client 3

Si verificano anomalie?

Domanda 5 (20%)
Illustrare il protocollo del commit a due fasi per l'organizzazione di un viaggio di lavoro, per il quale sia necessario (1) concordare la disponibilità di una persona da incontrare, (2) acquistare un biglietto aereo e (3) prenotare un albergo. Supporre che, per il biglietto aereo, sia possibile avere la conferma solo pagando e senza possibilità poi di annullare. Chiarire quali implicazioni ha quest'ultima caratteristica e quali condizioni debbono essere assunte, di conseguenza, riguardo alla persona da incontrare e all'albergo (se si vuole avere un comportamento analogo a quello del commit a due fasi).

Basi di dati II — 21 febbraio 2017
Cenni sulle soluzioni

Tempo a disposizione: un'ora e quarantacinque minuti.

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

Domanda 1 (15%)
Considerare un sistema con dischi con $N = 1000$ blocchi per traccia

- tempo medio di posizionamento della testina (tempo di seek) $t_S = 6$ msec
- tempo medio di latenza (attesa dovuta alla rotazione) $t_L = 2$ msec
- tempo minimo di lettura di un blocco $t_B = 10$ μsec

Rispondere alle seguenti domande mostrando formula e valore numerico

1. Qual è il tempo medio necessario per leggere un blocco del quale sia dato l'indirizzo fisico?

La somma del tempo medio di seek, del tempo medio di latenza e del tempo minimo di lettura di un blocco

$$t_{tot} = t_S + t_L + t_B = 6 + 2 + 0,01 \text{ msec } = \text{ ca } 8 \text{ msec}$$

2. Qual è il tempo medio necessario per la scansione sequenziale di un file costituito da $F = 200$ blocchi contigui, non letti di recente?

Il tempo medio necessario per leggere un blocco (il primo) più il tempo minimo di lettura per ciascuno degli altri

$$t_{tot} + (F - 1) \times t_B = \text{ ca } 9 \text{ msec}$$

3. Qual è il tempo che si può ipotizzare necessario per eseguire $m = 20.000$ accessi diretti in tempi ravvicinati a record di un file (molto grande) attraverso un indice che abbia profondità $p = 4$, fan-out $f_I = 50$, con disponibilità di circa $P = 4000$ pagine di buffer?

Si può immaginare che la radice e altri due livelli dell'indice restino nel buffer, dopo il primo caricamento, quindi, per ogni record, un accesso all'indice e uno al file, in posizioni non prevedibili (qualche accesso in più per il caricamento iniziale e qualcuno in meno per blocchi acceduti piè volte):

$$m \times ((p - 3) + 1) \times t_{tot} = \text{ ca } 320 \text{ sec}$$

4. Qual è il tempo che si può ipotizzare necessario per eseguire tre accessi diretti a record di un file attraverso un indice che abbia profondità $p = 4$ e fan-out (fattore di blocco dell'indice) $f_I = 50$, non usato di recente?

Si può immaginare che al massimo la radice dell'indice si trovi nel buffer, ma non gli altri nodi. Quindi, quattro accessi per l'indice e uno per il record del file, in posizioni non prevedibili:

$$3 \times ((p - 1) + 1) \times t_{tot} = \text{ ca } 40 \text{ msec}$$

Basi di dati II — 21 febbraio 2017

Domanda 2 (15%)
Si consideri un DBMS che preveda, in aggiunta alle tradizionali operazioni di lettura e scrittura, anche un'operazione INCREMENTA*(x,i)* che modifica il valore di x (intero) incrementandolo di i, senza renderlo 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 inc-lock, con la seguente regola di compatibilità (in aggiunta a quelle note per r-lock e w-lock): due inc-lock su un oggetto sono fra loro compatibili, mentre un inc-lock è incompatibile sia con un r-lock sia con un w-lock.

1. Spiegare intuitivamente perché questa tecnica può generare transazioni a maggiore livello di concorrenza (cioè con un throughput maggiore).

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

2. Dimostrare (almeno intuitivamente) che schedule 2-PL che soddisfino le regole di compatibilità note estese con quelle sopra citate risultano view-serializzabili.

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

Basi di dati II — 21 febbraio 2017

Domanda 3 (25%) Sia data una relazione $R(\underline{A}, B, C)$ contenente circa $N = 10.000.000$ ennuple di $r = 20$ byte ciascuna, di cui $k = 4$ per la chiave A, che contiene valori interi quasi consecutivi, da 1 a poco più di 10.000.000. Supporre che i blocchi abbiano dimensione $B = 2$KB, approssimabile come 2.000, che i puntatori ai record abbiano lunghezza $p = 6$; e che i nodi intermedi degli indici possano essere contenuti nei buffer.
Indicare il costo prevedibile per le seguenti operazioni

1. `SELECT * FROM R WHERE A >= 1000 AND A <=2000`
2. `SELECT COUNT(*) FROM R WHERE A >= 1000 AND A <=2000`
3. `SELECT * FROM R WHERE A = 1500`

in ciascuno dei seguenti casi:

(a) indice primario (sparso) su A realizzato con B+-tree;
(b) indice secondario (denso) su A realizzato con B+-tree.

Riportare le risposte nella tabella sottostante, indicando formula e valore numerico (con brevissimo commento, se necessario)

Possibile soluzione

Indichiamo con

- $F = B/r = 100$ il fattore di blocco del file
- $F_I = B/(a+p) = 200$ il fattore bi blocco massimo dell'indice; trattandosi di B+-tree, quello reale F'_I sarà minore (assumiamo del 30%) $F'_I = 150$ circa
- $N = 2000$ (o poco meno) il numero di record con valore di A compreso fra 1000 e 3000

(a)1 : poiché il file è ordinato, gli N record da trovare si trovano in $N/F = 20$ blocchi, che sono quindi accessibili (visto che l'indice è sparso), attraverso $\lceil (N/F)/F'_I \rceil = 1$ blocchi dell'indice; il costo è pari all'accesso alla foglia dell'indice (i livelli più alti sono nel buffer, quindi non costano niente) più a quelli dei record: $\lceil (N/F)/F'_I \rceil + N/F =$ circa 20
(b)1 : il file non è ordinato, i record sono sparpagliati, vanno acceduti a uno a uno e quindi servono N accessi più le foglie dell'indice (che è denso) $\lceil N/F'_I \rceil =$ circa 15 quindi trascurabile; costo totale circa $N = 2000$
(a)2 : come a(1) (l'indice è sparso e quindi per contare è necessario accedere ai record)
(b)2 : l'indice è denso, bastano le foglie dell'indice: $\lceil N/F'_I \rceil =$ circa 15
(a)3 : accesso diretto: una foglia dell'indice più un blocco del file, totale 2
(b)3 : come (a)3

Basi di dati II — 21 febbraio 2017

Domanda 4 (25%)
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 + 20 write(x) commit	
x = x + 10 write(x) commit		
		read(x) commit

Considerare uno scheduler con controllo di concorrenza basato su **Multversioni** (come in Postgres) e livello di isolamento **SERIALIZABLE** sulle prime due transazioni e **READ COMMITTED** sulla terza. Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia **1000**. 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 100*
begin read(x) *legge 100*		
	begin read(x) *legge 100* x = x + 20 write(x) *scrive 120* commit	
x = x + 10 write(x) *scrive 110* abort		
begin read(x) *legge 120* x = x + 10 write(x) *scrive 130* commit		
		read(x) *legge 130* commit

Si verificano anomalie?
Lettura inconsistente per il client 3

Basi di dati II — 21 febbraio 2017

Considerare nuovamente lo scenario della pagina precedente, ripetuto qui sotto per comodità.

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

Considerare uno scheduler con controllo di concorrenza ancora basato su **Multversioni** (come in Postgres) ma con livello di isolamento **READ COMMITTED** sulle prime due transazioni e **SERIALIZABLE** sulla terza. Mostrare il comportamento dello scheduler, supponendo che il valore iniziale dell'oggetto x sia ancora **1000**.

client 1	client 2	client 3
		begin read(x) *legge 100*
begin read(x) *legge 100*		
	begin read(x) *legge 100* x = x + 20 write(x) *scrive 120* commit	
x = x + 10 write(x) *scrive 110* commit		
		read(x) *legge 100* commit

Si verificano anomalie?
Perdita di aggiornamento fra il client 1 e il client 2

Basi di dati II — 21 febbraio 2017

Domanda 5 (20%)
Illustrare il protocollo del commit a due fasi per l'organizzazione di un viaggio di lavoro, per il quale sia necessario (1) concordare la disponibilità di una persona da incontrare, (2) acquistare un biglietto aereo e (3) prenotare un albergo. Supporre che, per il biglietto aereo, sia possibile avere la conferma solo pagando e senza possibilità poi di annullare. Chiarire quali implicazioni ha quest'ultima caratteristica e quali condizioni debbono essere assunte, di conseguenza, riguardo alla persona da incontrare e all'albergo (se si vuole avere un comportamento analogo a quello del commit a due fasi).

L'interessato è il coordinatore; per eseguire le operazioni in modo atomico è necessario che l'acquisto del biglietto aereo, che non è annullabile, venga effettuato per ultimo