Basi di dati II — Prova parziale — 14 maggio 2018 — Compito A

Tempo a disposizione: un'ora.

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

Basi di dati II — 14 maggio 2018 — Compito A

Domanda 1 (20%)

Considerare lo scenario a fianco in cui tre client diversi inviano richieste ad un gestore della 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 al 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). 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 **300** e quello dell'oggetto y sia **500**. 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)
begin read(x) x = x + 10 write(x) read(y)		
	begin read(y) y = y + 20 write(y) read(x) x = x – 20 write(x) commit	
y = y – 10 write(y) commit		
		(dopo molto tempo) read(x) commit

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — 14 maggio 2018 — Compito A

Domanda 2 (20%)

Considerare nuovamente lo scenario della domanda precedente, ripetuto qui a fianco per comodità. Considerare uno scheduler con controllo di concorrenza basato su **2PL stretto** con livello di isolamento **READ COMMITTED** sulle prime due transazioni e **SERIALIZABLE** sulla terza. Mostrare il comportamento dello scheduler, supponendo ancora che il valore iniziale dell'oggetto x sia **300** e quello dell'oggetto y sia **500**.

client 1	client 2	client 3
		begin read(x)
begin read(x) x = x + 10 write(x) read(y)		
	begin read(y) y = y + 20 write(y) read(x) x = x – 20 write(x) commit	
y = y – 10 write(y) commit		
		(dopo molto tempo) read(x) commit

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — 14 maggio 2018 — Compito A

Domanda 3 (20%) Considerare un sistema distribuito su cui viene eseguita una transazione che coinvolge tre nodi, un coordinatore M e due partecipanti N1 e N2. Dopo la richiesta di `prepare` da parte del coordinatore M, i due partecipanti ricevono e rispondono correttamente, e uno dei due, N2, va in crash subito dopo aver risposto. Il coordinatore riceve le risposte, prende la decisione, invia i relativi messaggi e subito dopo va anch'esso in crash (quindi senza fare in tempo a ricevere le conferme). Indicare, nello schema sottostante, una possibile sequenza di scritture sui log e invio di messaggi, supponendo che entrambi i nodi siano ripristinati abbastanza presto (ma che il coordinatore perda i messaggi di risposta inviati ad esso a seguito del commit). 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`→`N1,N2`). Supporre che nel log del coordinatore si scrivano solo i record di `prepare`, `commit` e `complete`, mentre i messaggi sono gestiti in memoria.

Nodo M		Nodo N1		Nodo N2	
Log	Messaggi	Log	Messaggi	Log	Messaggi
`prepare(N1,N2)`	`prepare`→`N1,N2`				
				crash	
crash					
restart					
				restart	

Eventuali commenti:

Basi di dati II — 14 maggio 2018 — Compito A

Domanda 4 (20%)
Per ciascuno degli schedule sotto riportati, indicare, scrivendo **sì** o **no** nelle varie caselle, a quali classi appartiene: S (seriale, rispetto a letture e scritture, ignorare commit e abort), CSR (conflict-serializzabile), S2PL (cioè generabile da uno scheduler basato su 2PL stretto), MV (cioè generabile da uno scheduler multiversion con controllo di serializzabilitï¿½: "a serializable transaction cannot modify or lock rows changed by other transactions after the serializable transaction began"). Negli schedule, s_i indica l'inizio della transazione i e c_i il suo commit.

	S	CSR	S2PL	MV
$s_2, s_1, r_2(x), w_2(x), r_1(x), w_1(x), c_2, c_1$				
$s_2, r_2(x), w_2(x), c_2, s_1, r_1(x), w_1(x), c_1$				
$s_1, r_1(x), s_2, r_2(x), w_1(x), c_1, w_2(x), c_2$				
$s_1, r_1(x), s_2, r_2(x), w_2(x), r_2(y), w_2(y), c_2, r_1(y), c_1$				

Domanda 5 (20%)
Considerare un sistema che utilizzi blocchi di lunghezza $D = 4$ KB (approssimabili a 4000 byte) e una tabella R con una struttura fisica heap con record a lunghezza fissa che occupano $L = 20$ byte ciascuno, in cui vengono inserite $M = 100.000$ ennuple, con valori della chiave tutti diversi fra loro e da quelli già nella relazione (quindi il sistema verifica il soddisfacimento del vincolo di chiave e ammette tutte le operazioni).
Rispondere alle domande seguenti, indicando formule e valori numerici:

Indicare il numero di scritture in memoria secondaria necessarie per realizzare i 100.000 inserimenti, supponendo che i record di log abbiano una lunghezza pari a circa il doppio di quella dei record del file, con riferimento ad un programma che utilizzi una transazione separata per ciascun inserimento

- numero di scritture di pagine di log:

- numero di scritture di pagine della relazione, nel caso si utilizzi una strategia undo-redo senza vincoli particolari

Come nel caso precedente, ma con riferimento ad un programma che, per realizzare i 100.000 inserimenti, utilizzi complessivamente $k = 10$ transazioni, ognuna con 10.000 inserimenti (e supponendo che non vi siano altre transazioni attive)

- numero di scritture di pagine di log:

- numero di scritture di pagine della relazione, sempre nel caso di strategia undo-redo senza vincoli particolari

Basi di dati II — Prova parziale — 14 maggio 2018 — Compito B

Tempo a disposizione: un'ora.

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

Basi di dati II — 14 maggio 2018 — Compito B

Domanda 1 (20%)

Considerare lo scenario a fianco in cui tre client diversi inviano richieste ad un gestore della 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 al 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). Considerare uno scheduler con controllo di concorrenza basato su **Multversioni** (come in Postgres) e 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 **300** e quello dell'oggetto y sia **700**. 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)
begin read(x) x = x + 10 write(x) read(y)		
	begin read(y) y = y + 20 write(y) read(x) x = x – 20 write(x) commit	
y = y – 10 write(y) commit		
		(dopo molto tempo) read(x) commit

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — 14 maggio 2018 — Compito B

Domanda 2 (20%)

Considerare nuovamente lo scenario della domanda precedente, ripetuto qui a fianco per comodità. Considerare uno scheduler con controllo di concorrenza basato su **2PL stretto** con livello di isolamento **SERIALIZABLE** sulle prime due transazioni e **READ COMMITTED** sulla terza. Mostrare il comportamento dello scheduler, supponendo ancora che il valore iniziale dell'oggetto x sia **300** e quello dell'oggetto y sia **700**.

client 1	client 2	client 3
		begin read(x)
begin read(x) x = x + 10 write(x) read(y)		
	begin read(y) y = y + 20 write(y) read(x) x = x – 20 write(x) commit	
y = y – 10 write(y) commit		
		(dopo molto tempo) read(x) commit

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — 14 maggio 2018 — Compito B

Domanda 3 (20%) Considerare un sistema distribuito su cui viene eseguita una transazione che coinvolge tre nodi, un coordinatore M e due partecipanti R1 e R2. Dopo la richiesta di `prepare` da parte del coordinatore M, i due partecipanti ricevono e rispondono correttamente, e uno dei due, R2, va in crash subito dopo aver risposto. Il coordinatore riceve le risposte, prende la decisione, invia i relativi messaggi e subito dopo va anch'esso in crash (quindi senza fare in tempo a ricevere le conferme). Indicare, nello schema sottostante, una possibile sequenza di scritture sui log e invio di messaggi, supponendo che entrambi i nodi siano ripristinati abbastanza presto (ma che il coordinatore perda i messaggi di risposta inviati ad esso a seguito del commit). 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`→`R1,R2`). Supporre che nel log del coordinatore si scrivano solo i record di `prepare`, `commit` e `complete`, mentre i messaggi sono gestiti in memoria.

Nodo M		Nodo R1		Nodo R2	
Log	Messaggi	Log	Messaggi	Log	Messaggi
`prepare(R1,R2)`	`prepare`→`R1,R2`				
				crash	
crash					
restart					
				restart	

Eventuali commenti:

Basi di dati II — 14 maggio 2018 — Compito B

Domanda 4 (20%)
Per ciascuno degli schedule sotto riportati, indicare, scrivendo **sì** o **no** nelle varie caselle, a quali classi appartiene: S (seriale, rispetto a letture e scritture, ignorare commit e abort), CSR (conflict-serializzabile), S2PL (cioè generabile da uno scheduler basato su 2PL stretto), MV (cioè generabile da uno scheduler multiversion con controllo di serializzabilitï¿½: "a serializable transaction cannot modify or lock rows changed by other transactions after the serializable transaction began"). Negli schedule, s_i indica l'inizio della transazione i e c_i il suo commit.

	S	CSR	S2PL	MV
$s_2, r_2(x), w_2(x), c_2, s_1, r_1(x), w_1(x), c_1$				
$s_1, r_1(x), s_2, r_2(x), w_1(x), c_1, w_2(x), c_2$				
$s_2, s_1, r_2(x), w_2(x), r_1(x), w_1(x), c_2, c_1$				
$s_1, r_1(x), s_2, r_2(x), w_2(x), r_2(y), w_2(y), c_2, r_1(y), c_1$				

Domanda 5 (20%)
Considerare un sistema che utilizzi blocchi di lunghezza $D = 8$ KB (approssimabili a 8000 byte) e una tabella R con una struttura fisica heap con record a lunghezza fissa che occupano $L = 20$ byte ciascuno, in cui vengono inserite $T = 50.000$ ennuple, con valori della chiave tutti diversi fra loro e da quelli già nella relazione (quindi il sistema verifica il soddisfacimento del vincolo di chiave e ammette tutte le operazioni).
Rispondere alle domande seguenti, indicando formule e valori numerici:

Indicare il numero di scritture in memoria secondaria necessarie per realizzare i 50.000 inserimenti, supponendo che i record di log abbiano una lunghezza pari a circa il doppio di quella dei record del file, con riferimento ad un programma che utilizzi una transazione separata per ciascun inserimento

- numero di scritture di pagine di log:

- numero di scritture di pagine della relazione, nel caso si utilizzi una strategia undo-redo senza vincoli particolari

Come nel caso precedente, ma con riferimento ad un programma che, per realizzare i 50.000 inserimenti, utilizzi complessivamente $k = 10$ transazioni, ognuna con 5000 inserimenti (e supponendo che non vi siano altre transazioni attive)

- numero di scritture di pagine di log:

- numero di scritture di pagine della relazione, sempre nel caso di strategia undo-redo senza vincoli particolari

Basi di dati II — Prova parziale — 14 maggio 2018 — Compito C

Tempo a disposizione: un'ora.

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

Basi di dati II — 14 maggio 2018 — Compito C

Domanda 1 (20%)

Considerare lo scenario a fianco in cui tre client diversi inviano richieste ad un gestore della 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 al 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). 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 **300** e quello dell'oggetto y sia **500**. 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)
begin read(x) x = x + 10 write(x) read(y)		
	begin read(y) y = y + 20 write(y) read(x) x = x – 20 write(x) commit	
y = y – 10 write(y) commit		
		(dopo molto tempo) read(x) commit

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — 14 maggio 2018 — Compito C

Domanda 2 (20%)

Considerare nuovamente lo scenario della domanda precedente, ripetuto qui a fianco per comodità. Considerare uno scheduler con controllo di concorrenza basato su **2PL stretto** con livello di isolamento **READ COMMITTED** sulle prime due transazioni e **SERIALIZABLE** sulla terza. Mostrare il comportamento dello scheduler, supponendo ancora che il valore iniziale dell'oggetto x sia **300** e quello dell'oggetto y sia **500**.

client 1	client 2	client 3
		begin read(x)
begin read(x) x = x + 10 write(x) read(y)		
	begin read(y) y = y + 20 write(y) read(x) x = x – 20 write(x) commit	
y = y – 10 write(y) commit		
		(dopo molto tempo) read(x) commit

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — 14 maggio 2018 — Compito C

Domanda 3 (20%) Considerare un sistema distribuito su cui viene eseguita una transazione che coinvolge tre nodi, un coordinatore C e due partecipanti P1 e P2. Dopo la richiesta di `prepare` da parte del coordinatore C, i due partecipanti ricevono e rispondono correttamente, e uno dei due, P2, va in crash subito dopo aver risposto. Il coordinatore riceve le risposte, prende la decisione, invia i relativi messaggi e subito dopo va anch'esso in crash (quindi senza fare in tempo a ricevere le conferme). Indicare, nello schema sottostante, una possibile sequenza di scritture sui log e invio di messaggi, supponendo che entrambi i nodi siano ripristinati abbastanza presto (ma che il coordinatore perda i messaggi di risposta inviati ad esso a seguito del commit). 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`→`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.

Nodo C		Nodo P1		Nodo P2	
Log	Messaggi	Log	Messaggi	Log	Messaggi
`prepare(P1,P2)`	`prepare`→`P1,P2`				
				crash	
crash					
restart					
				restart	

Eventuali commenti:

Basi di dati II — 14 maggio 2018 — Compito C

Domanda 4 (20%)
Per ciascuno degli schedule sotto riportati, indicare, scrivendo **sì** o **no** nelle varie caselle, a quali classi appartiene: S (seriale, rispetto a letture e scritture, ignorare commit e abort), CSR (conflict-serializzabile), S2PL (cioè generabile da uno scheduler basato su 2PL stretto), MV (cioè generabile da uno scheduler multiversion con controllo di serializzabilitï¿½: "a serializable transaction cannot modify or lock rows changed by other transactions after the serializable transaction began"). Negli schedule, s_i indica l'inizio della transazione i e c_i il suo commit.

	S	CSR	S2PL	MV
$s_2, s_1, r_2(x), w_2(x), r_1(x), w_1(x), c_2, c_1$				
$s_2, r_2(x), w_2(x), c_2, s_1, r_1(x), w_1(x), c_1$				
$s_1, r_1(x), s_2, r_2(x), w_2(x), r_2(y), w_2(y), c_2, r_1(y), c_1$				
$s_1, r_1(x), s_2, r_2(x), w_1(x), c_1, w_2(x), c_2$				

Domanda 5 (20%)
Considerare un sistema che utilizzi blocchi di lunghezza $D = 4$ KB (approssimabili a 4000 byte) e una tabella R con una struttura fisica heap con record a lunghezza fissa che occupano $L = 20$ byte ciascuno, in cui vengono inserite $N = 25.000$ ennuple, con valori della chiave tutti diversi fra loro e da quelli già nella relazione (quindi il sistema verifica il soddisfacimento del vincolo di chiave e ammette tutte le operazioni).
Rispondere alle domande seguenti, indicando formule e valori numerici:

Indicare il numero di scritture in memoria secondaria necessarie per realizzare i 25.000 inserimenti, supponendo che i record di log abbiano una lunghezza pari a circa il doppio di quella dei record del file, con riferimento ad un programma che utilizzi una transazione separata per ciascun inserimento

- numero di scritture di pagine di log:

- numero di scritture di pagine della relazione, nel caso si utilizzi una strategia undo-redo senza vincoli particolari

Come nel caso precedente, ma con riferimento ad un programma che, per realizzare i 25.000 inserimenti, utilizzi complessivamente $k = 5$ transazioni, ognuna con 5000 inserimenti (e supponendo che non vi siano altre transazioni attive)

- numero di scritture di pagine di log:

- numero di scritture di pagine della relazione, sempre nel caso di strategia undo-redo senza vincoli particolari

Basi di dati II — Prova parziale — 14 maggio 2018 — Compito D

Tempo a disposizione: un'ora.

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

Basi di dati II — 14 maggio 2018 — Compito D

Domanda 1 (20%)

Considerare lo scenario a fianco in cui tre client diversi inviano richieste ad un gestore della 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 al 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). Considerare uno scheduler con controllo di concorrenza basato su **Multversioni** (come in Postgres) e 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 **300** e quello dell'oggetto y sia **700**. 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)
begin read(x) x = x + 10 write(x) read(y)		
	begin read(y) y = y + 20 write(y) read(x) x = x – 20 write(x) commit	
y = y – 10 write(y) commit		
		(dopo molto tempo) read(x) commit

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — 14 maggio 2018 — Compito D

Domanda 2 (20%)

Considerare nuovamente lo scenario della domanda precedente, ripetuto qui a fianco per comodità. Considerare uno scheduler con controllo di concorrenza basato su **2PL stretto** con livello di isolamento **SERIALIZABLE** sulle prime due transazioni e **READ COMMITTED** sulla terza. Mostrare il comportamento dello scheduler, supponendo ancora che il valore iniziale dell'oggetto x sia **300** e quello dell'oggetto y sia **700**.

client 1	client 2	client 3
		begin read(x)
begin read(x) x = x + 10 write(x) read(y)		
	begin read(y) y = y + 20 write(y) read(x) x = x – 20 write(x) commit	
y = y – 10 write(y) commit		
		(dopo molto tempo) read(x) commit

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — 14 maggio 2018 — Compito D

Domanda 3 (20%) Considerare un sistema distribuito su cui viene eseguita una transazione che coinvolge tre nodi, un coordinatore C e due partecipanti N1 e N2. Dopo la richiesta di `prepare` da parte del coordinatore C, i due partecipanti ricevono e rispondono correttamente, e uno dei due, N2, va in crash subito dopo aver risposto. Il coordinatore riceve le risposte, prende la decisione, invia i relativi messaggi e subito dopo va anch'esso in crash (quindi senza fare in tempo a ricevere le conferme). Indicare, nello schema sottostante, una possibile sequenza di scritture sui log e invio di messaggi, supponendo che entrambi i nodi siano ripristinati abbastanza presto (ma che il coordinatore perda i messaggi di risposta inviati ad esso a seguito del commit). 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`→`N1,N2`). Supporre che nel log del coordinatore si scrivano solo i record di `prepare`, `commit` e `complete`, mentre i messaggi sono gestiti in memoria.

Nodo C		Nodo N1		Nodo N2	
Log	Messaggi	Log	Messaggi	Log	Messaggi
`prepare(N1,N2)`	`prepare`→`N1,N2`				
				crash	
crash					
restart					
				restart	

Eventuali commenti:

Basi di dati II — 14 maggio 2018 — Compito D

Domanda 4 (20%)
Per ciascuno degli schedule sotto riportati, indicare, scrivendo **sì** o **no** nelle varie caselle, a quali classi appartiene: S (seriale, rispetto a letture e scritture, ignorare commit e abort), CSR (conflict-serializzabile), S2PL (cioè generabile da uno scheduler basato su 2PL stretto), MV (cioè generabile da uno scheduler multiversion con controllo di serializzabilitï¿½: "a serializable transaction cannot modify or lock rows changed by other transactions after the serializable transaction began"). Negli schedule, s_i indica l'inizio della transazione i e c_i il suo commit.

	S	CSR	S2PL	MV
$s_1, r_1(x), s_2, r_2(x), w_2(x), r_2(y), w_2(y), c_2, r_1(y), c_1$				
$s_1, r_1(x), s_2, r_2(x), w_1(x), c_1, w_2(x), c_2$				
$s_2, s_1, r_2(x), w_2(x), r_1(x), w_1(x), c_2, c_1$				
$s_2, r_2(x), w_2(x), c_2, s_1, r_1(x), w_1(x), c_1$				

Domanda 5 (20%)
Considerare un sistema che utilizzi blocchi di lunghezza $D = 8$ KB (approssimabili a 8000 byte) e una tabella R con una struttura fisica heap con record a lunghezza fissa che occupano $L = 20$ byte ciascuno, in cui vengono inserite $R = 50.000$ ennuple, con valori della chiave tutti diversi fra loro e da quelli già nella relazione (quindi il sistema verifica il soddisfacimento del vincolo di chiave e ammette tutte le operazioni).
Rispondere alle domande seguenti, indicando formule e valori numerici:

Indicare il numero di scritture in memoria secondaria necessarie per realizzare i 50.000 inserimenti, supponendo che i record di log abbiano una lunghezza pari a circa il doppio di quella dei record del file, con riferimento ad un programma che utilizzi una transazione separata per ciascun inserimento

- numero di scritture di pagine di log:

- numero di scritture di pagine della relazione, nel caso si utilizzi una strategia undo-redo senza vincoli particolari

Come nel caso precedente, ma con riferimento ad un programma che, per realizzare i 50.000 inserimenti, utilizzi complessivamente $k = 5$ transazioni, ognuna con 10.000 inserimenti (e supponendo che non vi siano altre transazioni attive)

- numero di scritture di pagine di log:

- numero di scritture di pagine della relazione, sempre nel caso di strategia undo-redo senza vincoli particolari

Basi di dati II — Prova parziale — 14 maggio 2018 — Compito A

Cenni sulle soluzioni

Tempo a disposizione: un'ora.

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

Basi di dati II — 14 maggio 2018 — Compito A

Domanda 1 (20%)

Considerare lo scenario a fianco in cui tre client diversi inviano richieste ad un gestore della 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 al 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). 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 **300** e quello dell'oggetto y sia **500**. 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)
begin		
read(x)		
x = x + 10		
write(x)		
read(y)		
	begin	
	read(y)	
	y = y + 20	
	write(y)	
	read(x)	
	x = x – 20	
	write(x)	
	commit	
y = y – 10		
write(y)		
commit		
		(dopo molto tempo)
		read(x)
		commit

client 1	client 2	client 3
		begin
		read(x) — legge 300
begin		
read(x) — legge 300		
x = x + 10		
write(x) — scrive 310		
read(y) — legge 400		
	begin	
	read(y) — legge 500	
	y = y + 20	
	write(y) — scrive 520	
	read(x) — legge 300	
	x = x - 20	
	wlock(x) — bloccata	
y = y - 10		
wlock(y) — bloccata		
	...	
	abort	
write(y) — scrive 490		
commit		
	begin	
	read(y) — legge 490	
	y = y + 20	
	write(y) — scrive 510	
	read(x) — legge 310	
	x = x - 20	
	write(x) — scrive 290	
	commit	
		read(x)
		commit
		read(x) — legge 290

Si verificano anomalie?
Lettura inconsistente per il client 3

Domanda 2 (20%)

Considerare nuovamente lo scenario della domanda precedente, ripetuto qui a fianco per comodità. Considerare uno scheduler con controllo di concorrenza basato su **2PL stretto** con livello di isolamento **READ COMMITTED** sulle prime due transazioni e **SERIALIZABLE** sulla terza. Mostrare il comportamento dello scheduler, supponendo ancora che il valore iniziale dell'oggetto x sia **300** e quello dell'oggetto y sia **500**.

client 1	client 2	client 3
		begin
		read(x)
begin		
read(x)		
x = x + 10		
write(x)		
read(y)		
	begin	
	read(y)	
	y = y + 20	
	write(y)	
	read(x)	
	x = x – 20	
	write(x)	
	commit	
y = y – 10		
write(y)		
commit		
		(dopo molto tempo)
		read(x)
		commit

client 1	client 2	client 3
		begin
		read(x) — legge 300
begin		
read(x) — legge 200		
x = x + 10		
wlock(x) — bloccata		
	begin	
	read(y) — legge 500	
	y = y + 20	
	write(y) — scrive 520	
	read(x) — legge 200	
	x = x -20 10	
	wlock(x) — bloccata	
	...	
abort		
(1 e 2 non riescono comunque ad andare avanti)		*(dopo molto tempo)*
		read(x) — legge 300
		commit
(avendo poi abortito in momenti diversi possono forse concludersi regolarmente)		

Si verificano anomalie?
No, almeno nella soluzione ipotizzata.

Basi di dati II — 14 maggio 2018 — Compito A

Domanda 3 (20%) Considerare un sistema distribuito su cui viene eseguita una transazione che coinvolge tre nodi, un coordinatore M e due partecipanti N1 e N2. Dopo la richiesta di `prepare` da parte del coordinatore M, i due partecipanti ricevono e rispondono correttamente, e uno dei due, N2, va in crash subito dopo aver risposto. Il coordinatore riceve le risposte, prende la decisione, invia i relativi messaggi e subito dopo va anch'esso in crash (quindi senza fare in tempo a ricevere le conferme). Indicare, nello schema sottostante, una possibile sequenza di scritture sui log e invio di messaggi, supponendo che entrambi i nodi siano ripristinati abbastanza presto (ma che il coordinatore perda i messaggi di risposta inviati ad esso a seguito del commit). 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`→`N1,N2`). Supporre che nel log del coordinatore si scrivano solo i record di `prepare`, `commit` e `complete`, mentre i messaggi sono gestiti in memoria.

Nodo M		Nodo N1		Nodo N2	
Log	Messaggi	Log	Messaggi	Log	Messaggi
`prepare(N1,N2)`					
	`prepare`→`N1,N2`				
		`ready`		`ready`	
			`ready`→`M`		`ready`→`M`
				crash	
`commit`					
	`commit`→`N1,N2`				
crash					
				`commit`	
					`ack`→`M`
restart					
	`commit`→`N1,N2`				
					`ack`→`M`
				restart	
	`commit`→`N1`				
		`commit`			
			`ack`→`M`		
`complete`					

Eventuali commenti:

Basi di dati II — 14 maggio 2018 — Compito A

Domanda 4 (20%)
Per ciascuno degli schedule sotto riportati, indicare, scrivendo **sì** o **no** nelle varie caselle, a quali classi appartiene: S (seriale, rispetto a letture e scritture, ignorare commit e abort), CSR (conflict-serializzabile), S2PL (cioè generabile da uno scheduler basato su 2PL stretto), MV (cioè generabile da uno scheduler multiversion con controllo di serializzabilitï¿½: "a serializable transaction cannot modify or lock rows changed by other transactions after the serializable transaction began"). Negli schedule, s_i indica l'inizio della transazione i e c_i il suo commit.

Soluzioni per il compito A

	S	CSR	S2PL	MV
$s_2, s_1, r_2(x), w_2(x), r_1(x), w_1(x), c_2, c_1$	sì	sì	sì	sì
$s_2, r_2(x), w_2(x), c_2, s_1, r_1(x), w_1(x), c_1$	no	no	no	no
$s_1, r_1(x), s_2, r_2(x), w_1(x), c_1, w_2(x), c_2$	sì	sì	sì	no
$s_1, r_1(x), s_2, r_2(x), w_2(x), r_2(y), w_2(y), c_2, r_1(y), c_1$	no	no	no	sì *

(*) in questo caso $r_1(y)$ legge il valore di y all'inizio della transazione 1, cioè prima della scrittura $w_2(y)$

Domanda 5 (20%)
Considerare un sistema che utilizzi blocchi di lunghezza $D = 4$ KB (approssimabili a 4000 byte) e una tabella R con una struttura fisica heap con record a lunghezza fissa che occupano $L = 20$ byte ciascuno, in cui vengono inserite $M = 100.000$ ennuple, con valori della chiave tutti diversi fra loro e da quelli già nella relazione (quindi il sistema verifica il soddisfacimento del vincolo di chiave e ammette tutte le operazioni).
Rispondere alle domande seguenti, indicando formule e valori numerici:

Indicare il numero di scritture in memoria secondaria necessarie per realizzare i 100.000 inserimenti, supponendo che i record di log abbiano una lunghezza pari a circa il doppio di quella dei record del file, con riferimento ad un programma che utilizzi una transazione separata per ciascun inserimento

- numero di scritture di pagine di log:
  almeno $M = 100.000$, una per transazione
- numero di scritture di pagine della relazione, nel caso si utilizzi una strategia undo-redo senza vincoli particolari al massimo una per transazione, $M = 100.000$; probabilmente di meno, anche solo $M/f = 500$, una per blocco (dove f indica il fattore di blocco $f = D/L = 200$ nei compiti A e C e 400 nei compiti B e D)

Come nel caso precedente, ma con riferimento ad un programma che, per realizzare i 100.000 inserimenti, utilizzi complessivamente $k = 10$ transazioni, ognuna con 10.000 inserimenti (e supponendo che non vi siano altre transazioni attive)

- numero di scritture di pagine di log:
  per ogni transazione si debbono scrivere le pagine di log corrispondenti. Ogni transazione scrive M/k ennuple e quindi le relative scritture su log occupano $M/k \times L \times 2 = 400$KB e cioè $M/k \times L \times 2 * 1/D =$ ca.100 blocchi. In totale, per $k = 10$ transazioni, circa $M \times L \times 2 \times 1/D =$ ca.1000 scritture
- numero di scritture di pagine della relazione, sempre nel caso di strategia undo-redo senza vincoli particolari non si può dire con precisione, anche solo $M/f = 500$, una per blocco