Basi di dati II — compito A — 19 settembre 2018

Tempo a disposizione: un'ora e quarantacinque minuti.

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

Basi di dati II — compito A — 19 settembre 2018

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)
		commit
	read(x)	
	x = x – 20	
	write(x)	
	commit	
y = y – 10		
write(y)		
commit		

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — compito A — 19 settembre 2018

Considerare nuovamente lo scenario mostrato in precedenza, 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 **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)
		commit
	read(x)	
	x = x – 20	
	write(x)	
	commit	
y = y – 10		
write(y)		
commit		

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — compito A — 19 settembre 2018

Domanda 2 (20%)
In Postgres (e, con sintassi diverse, negli altri sistemi) è possibile ordinare fisicamente una relazione con il comando `CLUSTER`. L'ordinamento viene specificato sulla base di un indice già definito sulla stessa relazione. L'ordinamento non viene però mantenuto (se non eseguendo nuovamente il comando `CLUSTER`). Dal manuale:

> CLUSTER -- cluster a table according to an index
>
> **Synopsis**
>
> ```
> CLUSTER [VERBOSE] table_name [ USING index_name ]
> CLUSTER [VERBOSE]
> ```
>
> **Description**
>
> `CLUSTER` instructs PostgreSQL to cluster the table specified by ***table_name*** based on the index specified by ***index_name***. The index must already have been defined on ***table_name***.
>
> When a table is clustered, it is physically reordered based on the index information. Clustering is a one-time operation: when the table is subsequently updated, the changes are not clustered. That is, no attempt is made to store new or updated rows according to their index order. (If one wishes, one can periodically recluster by issuing the command again. Also, setting the table's `fillfactor` storage parameter to less than 100% can aid in preserving cluster ordering during updates, since updated rows are kept on the same page if enough space is available there.)

Sia data una relazione $R(\underline{A}, B, C)$ contenente circa $L = 3.000.000$ ennuple di $r = 100$ Byte ciascuna, con $v = 30.000$ valori diversi per l'attributo C, uniformemente distribuiti (quindi si può supporre che per ogni valore di C ci siano $L/v = 100$ ennuple con tale valore). Supporre che i blocchi abbiano dimensione $B = 4$KB approssimabile come 4.000.

Considerare le operazioni sotto elencate e indicare, nei riquadri, i costi delle `SELECT`:

1. `CREATE INDEX RCIX ON R (C)` (creazione di un indice su C; supporre che abbia profondità $p = 4$)
2. `CLUSTER R USING RCIX` (ordinamento di R sulla base dell'indice e quindi sull'attributo C)
3. Altre operazioni che non utilizzano l'indice `RCIX` e non modificano `R`
4. `SELECT * FROM R WHERE C = 100`

5. inserimento di $L/10 = 300.000$ ennuple, in ordine casuale, con valori di C pure uniformemente distribuiti (quindi si può supporre che vengano inserite 10 ennuple per ogni valore
6. `SELECT * FROM R WHERE C = 100`

Domanda 3 (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 — compito A — 19 settembre 2018

Domanda 4 (20%)
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 3 buffer, con un fattore di blocco pari a 2 e quindi uno spazio occupato dalla relazione pari a 8 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 sette chiamate di `next()`.

Run su disco

Buffer

Record prodotti dalle prime sette `next()`

131	...
441	...
742	...
231	...
541	...
342	...
942	...
872	...
601	...
173	...
496	...
575	...
145	...
635	...
986	...
855	...

Basi di dati II — compito A — 19 settembre 2018

Domanda 5 (20%) Considerare uno schema dimensionale relativo a presenze e reti di giocatori di calcio, che utilizzi, come tabella dei fatti e come una delle dimensioni, relazioni come quelle qui schematizzate:

KGiocatore	KSquadra	KTorneo	KStagione	Presenze	Reti
301	201	401	23	...	...
301	202	402	28	...	...
302	201	401	30	...	...
302	203	403	22	...	...
...	...	...	...	...	...

KTorneo	Nome	...
401	Campionato	...
402	Coppa dei Campioni	...
403	Coppa Italia	...
...	...	...

Supporre che si presentino la seguente esigenza di modifica:

- I tornei possono cambiare nome nel tempo: per esempio, il torneo nella prima ennupla potrebbe ad un certo punto cambiare nome da “Campionato” in “Serie A”; interessano selezioni e aggregazioni relative alle presenze e alle reti tanto con riferimento al nome del torneo (distinguendo quindi lil“Campionato” dalla “Serie A”) quanto alla sua identità (“Campionato” e “Serie A” in questo senso avrebbero la stessa identità; allo scopo si usa un codice, ma gli analisti talvolta preferiscono usare il nome attuale del torneo, quindi nell’esempio “Serie A” anche per le vecchie edizioni). Le modifiche sono rare, ma è possibile che ci siano tornei con vari cambiamenti di nome. Mostrare modifiche alle relazioni (una o entrambe) che permettano di soddisfare questa esigenza (mostrare i dati, con riferimento a quelli presenti negli esempi sopra, aggiungendo nuovi dati ragionevoli, che permettano di comprendente le modifiche).

Basi di dati II — compito B — 19 settembre 2018

Tempo a disposizione: un'ora e quarantacinque minuti.

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

Basi di dati II — compito B — 19 settembre 2018

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)
		commit
	read(x)	
	x = x – 20	
	write(x)	
	commit	
y = y – 10		
write(y)		
commit		

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — compito B — 19 settembre 2018

Considerare nuovamente lo scenario mostrato in precedenza, 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 **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)
		commit
	read(x)	
	x = x – 20	
	write(x)	
	commit	
y = y – 10		
write(y)		
commit		

client 1	client 2	client 3

Si verificano anomalie?

Basi di dati II — compito B — 19 settembre 2018

Domanda 2 (20%)
In Postgres (e, con sintassi diverse, negli altri sistemi) è possibile ordinare fisicamente una relazione con il comando `CLUSTER`. L'ordinamento viene specificato sulla base di un indice già definito sulla stessa relazione. L'ordinamento non viene però mantenuto (se non eseguendo nuovamente il comando `CLUSTER`). Dal manuale:

> CLUSTER -- cluster a table according to an index
>
> **Synopsis**
>
> ```
> CLUSTER [VERBOSE] table_name [ USING index_name ]
> CLUSTER [VERBOSE]
> ```
>
> **Description**
>
> `CLUSTER` instructs PostgreSQL to cluster the table specified by ***table_name*** based on the index specified by ***index_name***. The index must already have been defined on ***table_name***.
>
> When a table is clustered, it is physically reordered based on the index information. Clustering is a one-time operation: when the table is subsequently updated, the changes are not clustered. That is, no attempt is made to store new or updated rows according to their index order. (If one wishes, one can periodically recluster by issuing the command again. Also, setting the table's `fillfactor` storage parameter to less than 100% can aid in preserving cluster ordering during updates, since updated rows are kept on the same page if enough space is available there.)

Sia data una relazione $R(\underline{A}, B, C)$ contenente circa $L = 2.000.000$ ennuple di $r = 100$ Byte ciascuna, con $c = 20.000$ valori diversi per l'attributo C, uniformemente distribuiti (quindi si può supporre che per ogni valore di C ci siano $L/c = 100$ ennuple con tale valore). Supporre che i blocchi abbiano dimensione $B = 4$KB approssimabile come 4.000.

Considerare le operazioni sotto elencate e indicare, nei riquadri, i costi delle `SELECT`:

1. `CREATE INDEX RCIX ON R (C)` (creazione di un indice su C; supporre che abbia profondità $p = 4$)
2. `CLUSTER R USING RCIX` (ordinamento di R sulla base dell'indice e quindi sull'attributo C)
3. Altre operazioni che non utilizzano l'indice `RCIX` e non modificano `R`
4. `SELECT * FROM R WHERE C = 100`

5. inserimento di $L/10 = 200.000$ ennuple, in ordine casuale, con valori di C pure uniformemente distribuiti (quindi si può supporre che vengano inserite 10 ennuple per ogni valore
6. `SELECT * FROM R WHERE C = 100`

Domanda 3 (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 — compito B — 19 settembre 2018

Domanda 4 (20%)
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 3 buffer, con un fattore di blocco pari a 2 e quindi uno spazio occupato dalla relazione pari a 8 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 sette chiamate di `next()`.

Run su disco

Buffer

Record prodotti dalle prime sette `next()`

141	...
451	...
752	...
241	...
551	...
352	...
942	...
872	...
601	...
173	...
496	...
575	...
145	...
635	...
986	...
855	...

Basi di dati II — compito B — 19 settembre 2018

Domanda 5 (20%) Considerare uno schema dimensionale relativo a presenze e reti di giocatori di calcio, che utilizzi, come tabella dei fatti e come una delle dimensioni, relazioni come quelle qui schematizzate:

KGiocatore	KSquadra	KTorneo	KStagione	Presenze	Reti
201	401	301	27	...	...
201	402	302	28	...	...
202	401	301	30	...	...
202	403	303	22	...	...
...	...	...	...	...	...

KTorneo	Nome	...
301	Campionato	...
302	Coppa dei Campioni	...
303	Coppa Italia	...
...	...	...

Supporre che si presentino la seguente esigenza di modifica:

- I tornei possono cambiare nome nel tempo: per esempio, il torneo nella prima ennupla potrebbe ad un certo punto cambiare nome da “Campionato” in “Serie A”; interessano selezioni e aggregazioni relative alle presenze e alle reti tanto con riferimento al nome del torneo (distinguendo quindi lil“Campionato” dalla “Serie A”) quanto alla sua identità (“Campionato” e “Serie A” in questo senso avrebbero la stessa identità; allo scopo si usa un codice, ma gli analisti talvolta preferiscono usare il nome attuale del torneo, quindi nell’esempio “Serie A” anche per le vecchie edizioni). Le modifiche sono rare, ma è possibile che ci siano tornei con vari cambiamenti di nome. Mostrare modifiche alle relazioni (una o entrambe) che permettano di soddisfare questa esigenza (mostrare i dati, con riferimento a quelli presenti negli esempi sopra, aggiungendo nuovi dati ragionevoli, che permettano di comprendente le modifiche).

Basi di dati II — compito A — 19 settembre 2018
Cenni sulle soluzioni

Tempo a disposizione: un'ora e quarantacinque minuti.

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

Basi di dati II — compito A — 19 settembre 2018

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)
		commit
	read(x)	
	x = x – 20	
	write(x)	
	commit	
y = y – 10		
write(y)		
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 500		
	begin	
	read(y) — legge 500	
	y = y + 20	
	write(y) — scrive 520	
		read(x) — legge 300
		commit
	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	

Si verificano anomalie?
No

Basi di dati II — compito A — 19 settembre 2018

Considerare nuovamente lo scenario mostrato in precedenza, 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 **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)
		commit
	read(x)	
	x = x – 20	
	write(x)	
	commit	
y = y – 10		
write(y)		
commit		

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

Si verificano anomalie?
Nella soluzione ipotizzata, lettura inconsistente per il client 3.

Basi di dati II — compito A — 19 settembre 2018

Domanda 2 (20%)
In Postgres (e, con sintassi diverse, negli altri sistemi) è possibile ordinare fisicamente una relazione con il comando `CLUSTER`. L'ordinamento viene specificato sulla base di un indice già definito sulla stessa relazione. L'ordinamento non viene però mantenuto (se non eseguendo nuovamente il comando `CLUSTER`). Dal manuale:

> CLUSTER -- cluster a table according to an index
>
> **Synopsis**
>
> ```
> CLUSTER [VERBOSE] table_name [ USING index_name ]
> CLUSTER [VERBOSE]
> ```
>
> **Description**
>
> `CLUSTER` instructs PostgreSQL to cluster the table specified by ***table_name*** based on the index specified by ***index_name***. The index must already have been defined on ***table_name***.
>
> When a table is clustered, it is physically reordered based on the index information. Clustering is a one-time operation: when the table is subsequently updated, the changes are not clustered. That is, no attempt is made to store new or updated rows according to their index order. (If one wishes, one can periodically recluster by issuing the command again. Also, setting the table's `fillfactor` storage parameter to less than 100% can aid in preserving cluster ordering during updates, since updated rows are kept on the same page if enough space is available there.)

Sia data una relazione $R(\underline{A}, B, C)$ contenente circa $L = 3.000.000$ ennuple di $r = 100$ Byte ciascuna, con $v = 30.000$ valori diversi per l'attributo C, uniformemente distribuiti (quindi si può supporre che per ogni valore di C ci siano $L/v = 100$ ennuple con tale valore). Supporre che i blocchi abbiano dimensione $B = 4$KB approssimabile come 4.000.
Considerare le operazioni sotto elencate e indicare, nei riquadri, i costi delle `SELECT`:

1. `CREATE INDEX RCIX ON R (C)` (creazione di un indice su C; supporre che abbia profondità $p = 4$)
2. `CLUSTER R USING RCIX` (ordinamento di R sulla base dell'indice e quindi sull'attributo C)
3. Altre operazioni che non utilizzano l'indice `RCIX` e non modificano `R`
4. `SELECT * FROM R WHERE C = 100`

I record con lo stesso valore di `C` sono $L/v = 100$ e si trovano in uno stesso blocco o in blocchi consecutivi. Visto che il fattore di blocco è $B/r = 80$, i blocchi che li contengono sono $(L/v)/(B/r)$, arrotondato per eccesso ed eventualmente aumentato di uno, quindi 2 o 3, e quindi serviranno 5-7 accessi ($p = 4$ o $(p - 1 = 4$ (supponendo la radice nel buffer) per l'indice più 2 o 3 per i blocchi). L'indice si visita una volta sola.

5. inserimento di $L/10 = 300.000$ ennuple, in ordine casuale, con valori di C pure uniformemente distribuiti (quindi si può supporre che vengano inserite 10 ennuple per ogni valore
6. `SELECT * FROM R WHERE C = 100`

I 10 nuovi record con `C` $= 100$ si troveranno presumibilmente in 10 blocchi diversi e quindi il numero di accessi sarà pari al valore indicato nella risposta al punto precedente più 10 e quindi circa 15. L'indice, come nel caso precedente, si visita una volta sola

Basi di dati II — compito A — 19 settembre 2018

Domanda 3 (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. Il coordinatore invia il messaggio di prepare alla persona da incontrare e all'labergo (cioè propone l'incontro e prenota l'albergo in modo che l'albergo confermi ma lui possa annullare. Se entrambi rispondono positivamente (ready, in termini di 2PC), allora prenota anche l'aereo, e se questa prenotazione va a buon fine, allora conferma all'altra persona e all'albergo (cioè esegue il commit)

Basi di dati II — compito A — 19 settembre 2018

Domanda 4 (20%)
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 3 buffer, con un fattore di blocco pari a 2 e quindi uno spazio occupato dalla relazione pari a 8 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 sette chiamate di `next()`.

131 ...
441 ...
742 ...
231 ...
541 ...
342 ...
942 ...
872 ...
601 ...
173 ...
496 ...
575 ...
145 ...
635 ...
986 ...
855 ...

Run su disco

131 ...
231 ...
342 ...
441 ...
541 ...
742 ...

173 ...
496 ...
575 ...
601 ...
872 ...
942 ...

145 ...
635 ...
855 ...
986 ...

Buffer

541	...
742	...

←

575	...
601	...

←

145	...
635	...

←

Record prodotti dalle prime sette `next()`

131 ...
145 ...
173 ...
231 ...
342 ...
441 ...
496 ...

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 tre.
Nella prima passata, si costruiscono quindi tre run ordinati, ciascuno a partire da tre blocchi del file originario. L'ordinamento di ciascun run può essere effettuato in memoria centrale, usando tre 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 tre 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.

Basi di dati II — compito A — 19 settembre 2018

Domanda 5 (20%) Considerare uno schema dimensionale relativo a presenze e reti di giocatori di calcio, che utilizzi, come tabella dei fatti e come una delle dimensioni, relazioni come quelle qui schematizzate:

KGiocatore	KSquadra	KTorneo	KStagione	Presenze	Reti
301	201	401	23	...	...
301	202	402	28	...	...
302	201	401	30	...	...
302	203	403	22	...	...
...	...	...	...	...	...

KTorneo	Nome	...
401	Campionato	...
402	Coppa dei Campioni	...
403	Coppa Italia	...
...	...	...

Supporre che si presentino la seguente esigenza di modifica:

- I tornei possono cambiare nome nel tempo: per esempio, il torneo nella prima ennupla potrebbe ad un certo punto cambiare nome da “Campionato” in “Serie A”; interessano selezioni e aggregazioni relative alle presenze e alle reti tanto con riferimento al nome del torneo (distinguendo quindi lil“Campionato” dalla “Serie A”) quanto alla sua identità (“Campionato” e “Serie A” in questo senso avrebbero la stessa identità; allo scopo si usa un codice, ma gli analisti talvolta preferiscono usare il nome attuale del torneo, quindi nell’esempio “Serie A” anche per le vecchie edizioni). Le modifiche sono rare, ma è possibile che ci siano tornei con vari cambiamenti di nome. Mostrare modifiche alle relazioni (una o entrambe) che permettano di soddisfare questa esigenza (mostrare i dati, con riferimento a quelli presenti negli esempi sopra, aggiungendo nuovi dati ragionevoli, che permettano di comprendente le modifiche).

La dimensione Torneo va gestita come una “slowly changing dimension”, con un doppio accorgimento:

- va aggiunta una ennupla per ogni nuova versione di torneo
- vanno aggiunti un attributo con il nome attuale anche per la versione precedente e uno con un codice

KTorneo	Codice	NomeAlMomento	NomeAttuale	...
401	T1	Campionato	Seria A	...
402	T2	Coppa dei Campioni	. Coppa dei Campioni	...
403	T3	Coppa Italia	Coppa Italia	...
...	...	...	...	...
409	T2	Serie A	Serie A	...