# Basi di dati II

## Prova parziale — 17 maggio 2021

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

**Basi di dati II — 17 maggio 2021**

**Domanda 1** (25%)
Considerare un sistema che utilizzi blocchi di lunghezza $D = 1$ KB (approssimabili a 1000 byte) e una tabella $T$ con una struttura fisica heap con record a lunghezza fissa che occupano $L = 10$ byte ciascuno, in cui vengono inserite $R = 30.000$ 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).
Considerare tre casi (in ciascuno dei quali assumere che non ci siano altre transazioni attive):

1. il programma utilizza $R$=30.000 transazioni, una per ciascun inserimento
2. il programma utilizza $k$=1000 transazioni, ciascuna con $R/k$=30 inserimenti
3. il programma utilizza un'unica transazione, con $R$=30.000 inserimenti

Supponendo che i record del log abbiano una lunghezza pari a circa il triplo di quella dei record del file, indicare per ciascuno dei due casi:

- numero di scritture di pagine di log
- il numero minimo di scritture di pagine dati, assumendo che il sistema utilizzi una strategia undo-redo senza vincoli particolari

Rispondere in modo sintetico, su sei righe separate (indicando numeri e possibilmente anche formule):
"1. pagine di log: ... "
"1. pagine dati: ..."
"2. pagine di log: ... "
"2. pagine dati: ..."
"3. pagine di log: ... "
"3. pagine dati: ..."

**Basi di dati II — 17 maggio 2021**

**Domanda 2** (25%)
Considerare una relazione definita con il seguente comando:

```
create table conti (num integer primary key, saldo integer not null);
```

Supporre che la relazione contenga almeno una ennupla e considerare il seguente scenario in cui due client inviano richieste ad un gestore del controllo di concorrenza. In sostanza, ciascuna delle due transazioni inserisce nella relazione `conti` una nuova ennupla, con valore della chiave maggiore (di una unità) del massimo valore presente e un opportuno valore per il saldo (1000 nella prima transazione e 1 nella seconda).
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. Supporre che, in caso di abort, le transazioni non vengano rilanciate.

| | |
|---|---|
| `start transaction isolation level serializable;`<br>`select max(num) as n into app1 from conti;`<br><br><br>`insert into conti (num, saldo)`<br>`  select n+1 as num, 1000 as saldo from app1;`<br><br><br>`commit ;` | <br><br><br>`start transaction isolation level serializable;`<br>`select max(num) as n into app2 from conti;`<br><br><br>`insert into conti (num, saldo)`<br>`  select n+1 as num, 1 as saldo from app2;`<br><br>`commit ;` |

1. Considerare uno scheduler con controllo di concorrenza basato su **Multversioni** (come in Postgres) e descrivere l'esito delle transazioni, indicando l'effetto di ciascuna azione.

2. Descrivere invece che cosa succede se entrambe le transazioni vengono eseguite con livello di isolamento `repeatable read`

Rispondere alle due domande nello spazio fornito da Moodle qui sotto, cercando di dare adeguate motivazioni.

**Basi di dati II — 17 maggio 2021**

**Domanda 3** (25%)
Per ciascuno degli schedule sotto riportati, indicare a quali classi appartiene: S (seriale, rispetto a letture e scritture, ignorare i commit), CSR (conflict-serializzabile), S2PL (cioè generabile da uno scheduler basato su 2PL stretto), MV (cioè generabile da uno scheduler multiversion con controllo di serializzabilità). Negli schedule, $c_i$ indica il commit della transazione $i$; l'inizio di una transazione è costituito dalla sua prima azione.

| | | S | CSR | S2PL | MV |
|---|---|---|---|---|---|
| 1. | $r_2(x), w_2(x), r_1(x), w_1(x), c_2, c_1$ | | | | |
| 2. | $r_2(x), w_2(x), r_1(z), c_2, r_1(x), w_1(x), c_1$ | | | | |
| 3. | $r_1(x), r_2(x), w_2(x), r_2(y), w_2(y), c_2, r_1(y), c_1$ | | | | |
| 4. | $r_1(x), r_2(x), w_1(x), c_1, w_2(x), c_2$ | | | | |

Riportare la risposta su quattro righe, con un elenco in ciascuna di "sì" o "no" corrispondenti alle caselle della tabella.

**Basi di dati II — 17 maggio 2021**

**Domanda 4** (25%)
Considerare un sistema distribuito su cui viene eseguita una transazione T che coinvolge un coordinatore C e due partecipanti R1 e R2. Come schematizzato sotto, coordinatore C invia il messaggio di `prepare` e subito dopo va in crash mentre i due partecipanti hanno il tempo di ricevere e rispondere correttamente, ma uno dei due, R1, va in crash poco dopo. Supporre che, dopo il crash, il coordinatore C segua la strategia più semplice, abortendo la transazione, e che C e R1 siano ripristinati abbastanza presto. Supporre che il coordinatore non scriva nel log quali messaggi di ack ha ricevuto.

| | Nodo C | | Nodo R1 | | Nodo R2 | |
|---|---|---|---|---|---|---|
| | Log | Messaggi | Log | Messaggi | Log | Messaggi |
| 1 | `prepare(T,R1,R2)` | | | | | |
| 2 | | `prepare(T)`→`R1,R2` | | | | |
| 3 | *crash* | | | | | |
| 4 | | | | | | |
| 5 | | | *crash* | | | |
| 6 | | | | | | |
| 7 | *restart* | | | | | |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |
| 12 | | | | | | |
| 13 | | | | | | |
| 14 | | | *restart* | | | |
| 15 | | | | | | |

Indicare quali record scrive nel proprio log ciascuno dei nodi e quali messaggi invia. Per comodità, è indicata a sinistra una numerazione degli istanti, utilizzabile nelle risposte. Ad esempio, per indicare che, all'istante 3, il nodo R1 registra nel log un record di `ready`, si scriva "3: R1 scrive nel log ready(T)"; oppure, per indicare che, all'istante 4, R1 manda un messaggio di `ready` al coordinatore, si scriva "4: R1 invia ready(T) a C ". Supporre che il timeout per ogni richiesta scatti dopo 3 o 4 istanti.

# Basi di dati II

## Prova parziale — 17 maggio 2021

## Cenni sulle soluzioni

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

**Basi di dati II — 17 maggio 2021**

**Domanda 1** (25%)
Considerare un sistema che utilizzi blocchi di lunghezza $D = 1$ KB (approssimabili a 1000 byte) e una tabella $T$ con una struttura fisica heap con record a lunghezza fissa che occupano $L = 10$ byte ciascuno, in cui vengono inserite $R = 30.000$ 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).
Considerare tre casi (in ciascuno dei quali assumere che non ci siano altre transazioni attive):

1. il programma utilizza $R$=30.000 transazioni, una per ciascun inserimento
2. il programma utilizza $k$=1000 transazioni, ciascuna con $R/k$=30 inserimenti
3. il programma utilizza un'unica transazione, con $R$=30.000 inserimenti

Supponendo che i record del log abbiano una lunghezza pari a circa il triplo di quella dei record del file, indicare per ciascuno dei due casi:

- numero di scritture di pagine di log
- il numero minimo di scritture di pagine dati, assumendo che il sistema utilizzi una strategia undo-redo senza vincoli particolari

Rispondere in modo sintetico, su sei righe separate (indicando numeri e possibilmente anche formule):
"1. pagine di log: ... "
"1. pagine dati: ..."
"2. pagine di log: ... "
"2. pagine dati: ..."
"3. pagine di log: ... "
"3. pagine dati: ..."

**caso 1. pagine di log** tante quante sono le transazioni, $R = 30.000$

**caso 1. pagine dati** : pari al numero di blocchi necessari, $R \times L/D = 300$

**caso 2. pagine di log** ancora una per transazione (deve scrivere $R/k$=30 record di circa $3 \times L = 30$ byte, quindi circa 900, che entrano in un blocco; quindi in totale $k$=1000 scritture; in effetti, poiché il log viene scritto in modo sequenziale e ogni blocco viene riempito, nella maggior parte dei casi la porzione da scrivere completa un blocco e inizia il successivo, quindi il numero di scritture sarà fra $k$=1000 e $2 \times k$=2000, ma tutte le risposte vanno bene

**caso 2. pagine dati** analogo al caso 1

**caso 3. pagine di log** la transazione è una sola, quindi è sufficiente calcolare lo spazio occupato quindi possiamo considerare semplicemente il numero di blocchi necessari: $3 \times (R \times L/D) = 900$ (cui andrebbe aggiunto lo spazio per record di commit, ma trascuriamo)

**caso 3. pagine dati** analogo al caso 1

**Basi di dati II — 17 maggio 2021**

**Domanda 2** (25%)
Considerare una relazione definita con il seguente comando:

```
create table conti (num integer primary key, saldo integer not null);
```

Supporre che la relazione contenga almeno una ennupla e considerare il seguente scenario in cui due client inviano richieste ad un gestore del controllo di concorrenza. In sostanza, ciascuna delle due transazioni inserisce nella relazione `conti` una nuova ennupla, con valore della chiave maggiore (di una unità) del massimo valore presente e un opportuno valore per il saldo (1000 nella prima transazione e 1 nella seconda).
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. Supporre che, in caso di abort, le transazioni non vengano rilanciate.

| | |
|---|---|
| `start transaction isolation level serializable;`<br>`select max(num) as n into app1 from conti;`<br><br>`insert into conti (num, saldo)`<br>`  select n+1 as num, 1000 as saldo from app1;`<br><br>`commit ;` | `start transaction isolation level serializable;`<br>`select max(num) as n into app2 from conti;`<br><br>`insert into conti (num, saldo)`<br>`  select n+1 as num, 1 as saldo from app2;`<br><br>`commit ;` |

1. Considerare uno scheduler con controllo di concorrenza basato su **Multversioni** (come in Postgres) e descrivere l'esito delle transazioni, indicando l'effetto di ciascuna azione.
2. Descrivere invece che cosa succede se entrambe le transazioni vengono eseguite con livello di isolamento `repeatable read`

Rispondere alle due domande nello spazio fornito da Moodle qui sotto, cercando di dare adeguate motivazioni.

Sarebbe stato utile riportare il dettaglio degli eventi.

1. La transazione del secondo client viene inizialmente bloccata perché cerca di accedere al dato inserito dalla transazione del primo client. Al commit della prima, la seconda non riesce ad avere il lock, per via del ciclo:
   `ERROR: could not serialize access due to read/write dependencies among transactions`
   `DETAIL: Reason code:  Canceled on identification as a pivot, during write.`
2. Il lock blocca la seconda transazione come nel caso precedente, poi ottiene il lock ma, quando prova a scrivere, va in errore per violazione del vincolo di chiave (entrambe le transazioni scrivono lo stesso valore "n+1")

**Basi di dati II — 17 maggio 2021**

**Domanda 3** (25%)
Per ciascuno degli schedule sotto riportati, indicare a quali classi appartiene: S (seriale, rispetto a letture e scritture, ignorare i commit), CSR (conflict-serializzabile), S2PL (cioè generabile da uno scheduler basato su 2PL stretto), MV (cioè generabile da uno scheduler multiversion con controllo di serializzabilità). Negli schedule, $c_i$ indica il commit della transazione $i$; l'inizio di una transazione è costituito dalla sua prima azione.

| | | S | CSR | S2PL | MV |
|---|---|---|---|---|---|
| 1. | $r_2(x), w_2(x), r_1(x), w_1(x), c_2, c_1$ | sì | sì | no | no |
| 2. | $r_2(x), w_2(x), r_1(z), c_2, r_1(x), w_1(x), c_1$ | sì | sì | sì | no |
| 3. | $r_1(x), r_2(x), w_2(x), r_2(y), w_2(y), c_2, r_1(y), c_1$ | no | no | no | sì |
| 4. | $r_1(x), r_2(x), w_1(x), c_1, w_2(x), c_2$ | no | no | no | no |

Riportare la risposta su quattro righe, con un elenco in ciascuna di "sì" o "no" corrispondenti alle caselle della tabella.

**Basi di dati II — 17 maggio 2021**

**Domanda 4** (25%)
Considerare un sistema distribuito su cui viene eseguita una transazione T che coinvolge un coordinatore C e due partecipanti R1 e R2. Come schematizzato sotto, coordinatore C invia il messaggio di `prepare` e subito dopo va in crash mentre i due partecipanti hanno il tempo di ricevere e rispondere correttamente, ma uno dei due, R1, va in crash poco dopo. Supporre che, dopo il crash, il coordinatore C segua la strategia più semplice, abortendo la transazione, e che C e R1 siano ripristinati abbastanza presto. Supporre che il coordinatore non scriva nel log quali messaggi di ack ha ricevuto.

| | Nodo C | | Nodo R1 | | Nodo R2 | |
|---|---|---|---|---|---|---|
| | Log | Messaggi | Log | Messaggi | Log | Messaggi |
| 1 | `prepare(T,R1,R2)` | | | | | |
| 2 | | `prepare(T)`→`R1,R2` | | | | |
| 3 | *crash* | | | | | |
| 4 | | | `ready` | | `ready` | |
| 5 | | | | *crash* | | |
| 6 | | | | | | |
| 7 | *restart* | | | | | |
| 8 | | | | | | |
| 9 | | | | | | |
| 10 | | | | | | |
| 11 | | | | | | |
| 12 | | | | | | |
| 13 | | | | | | |
| 14 | | | | *restart* | | |
| 15 | | | | | | |

Indicare quali record scrive nel proprio log ciascuno dei nodi e quali messaggi invia. Per comodità, è indicata a sinistra una numerazione degli istanti, utilizzabile nelle risposte. Ad esempio, per indicare che, all'istante 3, il nodo R1 registra nel log un record di `ready`, si scriva "3: R1 scrive nel log ready(T)"; oppure, per indicare che, all'istante 4, R1 manda un messaggio di `ready` al coordinatore, si scriva "4: R1 invia ready(T) a C ". Supporre che il timeout per ogni richiesta scatti dopo 3 o 4 istanti.

3. R1 scrive nel log ready(T), R2 scrive nel log ready(T)
4. R1 invia ready(T) a C, R2 invia ready(T) a C
8. C scrive nel log abort(T)
9. C invia abort(T) a R1 e R2
10. R2 scrive nel log abort(T)
11. R2 invia ack(T) a C
12. C invia abort(T) a R1
15. C invia abort(T) a R1
16. R1 scrive nel log abort(T)
17. R1 invia ack(T) a C
18. C scrive nel log complete(T)