IIC2413 – Bases de Datos

Guía – Transacciones y Logging

Preguntas

1. **Schedules.** Considere el schedule **S** de el Cuadro 1. Argumente lo siguiente:

   - **S** no es serial.
   - **S** no es conflict-serializable.
   - **S** es serializable.

T1	T2	T3
R(a)		
	W(a)	
	Commit	
W(a)		
Commit		
		W(a)
		R(b)
		Commit

Cuadro 1: Schedule **S**.

   Respuestas:

   - **S** no es serial.
     - Las operaciones de distintas transacciones están mezcladas. Por lo tanto el schedule no es serial.
   - **S** no es conflict-serializable.
     - Existe un ciclo en su grafo. En particulár, hay un arista desde `T1` y `T2` (R(a) en `T1` y W(a) en `T2`), y otra al revés (W(a) en `T2` y W(a) en `T1`).
   - **S** es serializable.
     - Dado qué `3` sobreescribe todo lo que hacen `T1` y `T2`, el schedule **S** es equivalente a `T1;T2;T3` y a `T2;T1;T3` (los dos son serial).

2. **Undo logging.** Suponga que su sistema tuvo una falla. Al reiniciar el sistema, el sistema se encuentra con el *log file* que se muestra a continuación, en la tabla "Log Undo". Suponiendo que la política de *recovery* es la de *Undo Logging*, indique:

   - Hasta qué parte del *log* debo leer.
   - Qué variables deben deshacer sus cambios y cuál es el valor con el que quedarán.
   - Qué variables (de las que aparecen en el *log*) no son cambiadas en el proceso.

Log Undo
<START T1>
<START T2>
<T1, a, 22>
<T2, b, 1>
<START T3>
<T2, b, 2>
<START T4>
<T4, c, 7>
<COMMIT T1>
<T3, d, 22>
<START CKPT (T2,T3,T4)>
<T3, a, 10>
<T2, b, 11>
<START T5>
<T5, d, 5>
<ABORT T4>
<T2, e, 32>
<COMMIT T2>
<T5, f, -3>
<COMMIT T5>

Respuestas:

- Hasta qué parte del *log* debo leer.
  - Cómo hay un `START CKPT (T2,T3,T4)` sin su `END CKPT` respectivo, leemos desde la transacción más antigua entre `T2,T3,T4` (en este caso `T2`).
- Qué variables deben deshacer sus cambios y cuál es el valor con el que quedarán.
  - Transacciones `T1,T2,T4` y `T5` están finalizadas. Por lo tanto, solo hay que hacer el `UNDO` de `T3`. Las únicas variables que cambiarán en este proceso son `d` y `a`.
- Qué variables (de las que aparecen en el *log*) no son cambiadas en el proceso.
  - El resto.

3. **Redo logging.** Considerando el *schedule*:

Log Redo
<START T1>
<T1, a, 1>
<COMMIT T1>
<START T2>
<T2, b, 2>
<T2, c, 3>
<COMMIT T2>
<START T3>
<END T1>
<T3, a, 10>
<START CKPT (T3)>
<T3, d, 23>
<START T4>
<END T2>
<END CKPT>
<COMMIT T3>
<T4, e, 11>

Indique:

- Desde qué parte del *log* debo comenzar el proceso de *redo*.
- Qué variables deben rehacer sus cambios y cuál es el valor con el que quedarán.
- Qué variables (de las que aparecen en el *log*) no son cambiadas en el proceso.
- Si no hubiesemos encontrado la línea `<END CKPT>`, ¿desde qué parte del *log* debería comenzar el proceso de *redo*?.

Respuestas:

- Desde qué parte del *log* debo comenzar el proceso de *redo*.
  - Dado que existe un `END CKPT`, debo leer hasta el `<START T3>`, que es la transacción más antigua que se señala en el `START CKPT`.
- Qué variables deben rehacer sus cambios y cuál es el valor con el que quedarán.
  - Tenemos la certeza de que T1 y T2 están guardadas en disco, porque tenemos la presencia de un `END CKPT`. Fuera de esas transacciones, debemos rehacer todas las otras transacciones que están marcadas con `COMMIT` y no tienen un `END`. En este caso es sólo T3. Por lo que `<T3, a, 10>` nos indica que debemos rehacer **a** al valor 10 y `<T3, d, 23>` nos indica que d debe rehacer **d** a 23.
- Qué variables (de las que aparecen en el *log*) no son cambiadas en el proceso.
  - Continuando con el proceso anterior, T4 no está marcado con commit, por lo que **e** no se debe tocar. T4 debe ser marcada con `ABORT` en el proceso. **b** y **c** tampoco son cambiadas en el proceso. Lo cambios efectuados a la variable **a** debido a T1 tampoco deben ser realizados de nuevo.
- Si no hubiesemos encontrado la línea `<END CKPT>`, ¿desde qué parte del *log* debería comenzar el proceso de *redo*?.
  - Debemos encontrar sí o sí un `<END CKPT>`. Así que como no hay otro se debe leer el *log* entero.