Parcialito IV: Concurrencia, Transacciones y Recuperación

Base de Datos 75.15/75.28/95.05 - Cátedra Román

Segundo Cuatrimestre 2025

Melanie García Lapegna - 111848

Concurrencia

Para las siguientes planificaciones:

A. Dibujar los grafos de precedencia

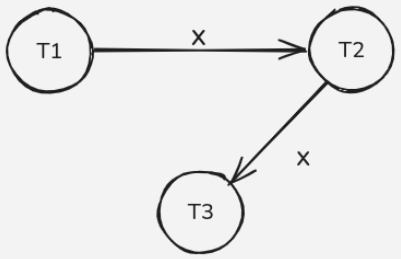
B. Cantidad y Listado los conflictos

C. Determinar cuáles son serializables (justificando correctamente en una oración).

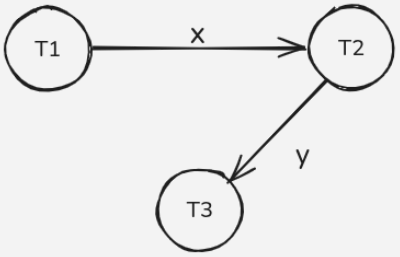
D. Es recuperable? (justificando correctamente en una oración).

E. En caso de no recuperable, que tendríamos que modificar para que sea recuperable? (justificando correctamente en una oración).

1. bT1; bT2; bT3; RT1(X); WT2(X); RT3(X); WT1(Y); RT2(Y); WT3(Z); cT1; cT2; cT3;

A. Grafo	
B. Cantidad total de conflictos	3
B. Listado de conflictos	$T_1 \rightarrow T_2$, $R_{T1}(X)$, $W_{T2}(X)$ $T_2 \rightarrow T_3$, $W_{T2}(X)$, $R_{T3}(X)$ $T_1 \rightarrow T_2$, $W_{T1}(Y)$, $R_{T2}(Y)$
C. Es Serializable?	Si, ya que su grafo de precedencia no tiene ciclos ∴ la ejecución de estas instrucciones deja a la bdd en un estado equivalente a uno que hubiera dejado alguna ejecución serial.
D. Es Recuperable?	Si, ya que todas las instrucciones de tipo READ que leen un ítem que fue previamente modificado por **otra transacción**, luego commitean posteriormente a la transacción que realizó la modificación.
E. Modificación para hacerlo recuperable	-

2. bT1; bT2; bT3; RT1(X); WT1(X); RT2(X); WT2(X); RT2(Y); WT2(Y); RT3(Y); WT3(Y); cT3; cT2; cT1;

A. Grafo	
B. Cantidad total de conflictos	6
B. Listado de conflictos	$T_1 \rightarrow T_2$, $R_{T1}(X)$, $W_{T2}(X)$ $T_1 \rightarrow T_2$, $W_{T1}(X)$, $R_{T2}(X)$ $T_1 \rightarrow T_2$, $W_{T1}(X)$, $W_{T2}(X)$ $T_1 \rightarrow T_2$, $W_{T1}(X)$, $R_{T2}(X)$ $T_2 \rightarrow T_3$, $W_{T2}(Y)$, $R_{T3}(Y)$ $T_2 \rightarrow T_3$, $W_{T2}(Y)$, $W_{T3}(Y)$
C. Es Serializable?	Si, ya que su grafo de precedencia no tiene ciclos .
D. Es Recuperable?	No, ya que tenemos $W_{T1}(X)$, $R_{T2}(X)$, luego c_{T2} , c_{T1} y $W_{T2}(Y)$, $R_{T3}(Y)$, luego c_{T3} , c_{T2}. Las transacciones que tienen una instrucción de lectura posterior a una modificación al mismo ítem por parte de otra transacción deben commitear luego de de esta transacción (de la que realizó la modificación) para ser recuperable.
E. Modificación para hacerlo recuperable	Para que sea recuperable los commits deberían hacerse en el siguiente orden c_{T1} , c_{T2} , c_{T3}.

Recuperación

Un SGBD implementa el algoritmo de recuperación **UNDO/REDO** con checkpoint activo.
Luego de una falla, el sistema encuentra el siguiente archivo de log incompleto:

Nro línea	Log
01	(BEGIN, T1);
02	(WRITE T1, A, 1, __);
03	(BEGIN CKPT, __);
04	(COMMIT, T1);
05	(BEGIN, T2);
06	(WRITE T2, A, 2, 4);
07	(COMMIT, T2);
08	(END CKPT);
09	(BEGIN, T3);
10	(BEGIN CKPT, __);
11	(WRITE T3, A, __ , 3);
12	(WRITE T3, B, 6, 7);

Complete los **valores faltantes** y explique **cómo se llevará a cabo** el procedimiento de recuperación, indicando **hasta qué punto del archivo** de log se deberá retroceder (justificando por qué), y **qué cambios** deberán ser realizados en disco y en el archivo de log.
Para la entrega, entregue el archivo de log completo, y en lo posible, ponga las partes faltantes en color ROJO.

El archivo log completo queda:

Nro_linea	Log
01	(BEGIN,T1)
02	(WRITE,T1,A,1,**2**)
03	(BEGIN CKPT,**T1**)
04	(COMMIT,T1)
05	(BEGIN,T2)
06	(WRITE,T2,A,2,4)
07	(COMMIT,T2)
08	(END CKPT)
09	(BEGIN,T3)
10	(BEGIN CKTP,**T3**)
11	(WRITE,T3,A,**4**,3)
12	(WRITE,T3,B,6,7)
13	**(ABORT,T3)**

Para la **recuperación** se comienza recorriendo el log de adelante hacia atrás (se comienza desde la línea 12) y se va hasta la línea 01 deshaciendo las instrucciones de transacciones que no fueron commitedas(UNDO).

Seguimiento y justificación de esta primera parte:

Obs: a medida que se recorre el log voy actualizando la lista de transacciones commitedas.

-L12 : hay un **WRITE** de T3 la cual no fue commiteada previamente ⇒ agrego a no commiteadas y deshago la instrucción.

CAMBIO EN DISCO B→6 (utilizo valor viejo)

-L11 : hay un **WRITE** de T3 la cual no fue commiteada previamente ⇒ deshago la instrucción.

CAMBIO EN DISCO A→4 (utilizo valor viejo)

-L10 : hay un **BEGIN CKPT**, pero como no llego a hacerse el **END CKPT** antes de la falla no nos sirve ya que no nos asegura que las instrucciones previas fueron flusheadas a disco.

-L09 : hay un **BEGIN** de **T3**, seguimos de largo.

-L08 : hay un **END CKPT**. Vemos que transacciones estaban activas en el **BEGIN CKPT** que le corresponde. En la línea 03 tenemos **BEGIN CKPT T1**, es decir, la única transacción activa en ese momento es la **T1.**
Por lo tanto, en esta primera etapa de UNDO debemos ir hasta el **BEGIN de T1** (si tuviéramos más operaciones por encima de esta, tendríamos certezas de que ya fueron escritas en el disco ya que pasamos por un **END CKPT** y además sabriamos que ya commitearon por lo que no haria falta ir más allá del BEGIN T1).
Debemos ir hasta el BEGIN de T1 porque T1 quizás nunca commiteo, por lo que podríamos tener que deshacer sus cambios.

-L07 : hay un **COMMIT T2** ⇒ agrego a T2 a commiteadas.

-L06 : hay un **WRITE de T2** , como esta commiteada no la deshago.

-L05 : hay un **BEGIN T2** , sigo.

-L04 : hay un **COMMIT T1** ⇒ agrego a T1 a commiteadas.

-L03 : hay un **BEGIN CKTP**, ya fue explicado previamente, pero todo lo que esté por encima de este, como ya pasamos por su END CKPT, sabemos que ya fue escrito en disco. Vamos hasta el **BEGIN** de la única transacción activa que tenía.

-L02 : hay un **WRITE de T1** , como esta commiteada no la deshago.

-L01 : hay un **BEGIN T1** , termine la etapa de UNDO.

Transacciones commiteadas	T2 , T1
Transacciones NO commiteadas	T3

Para esta segunda parte, se recorre al log de atrás hacia adelante, persistiendo los **WRITE** de las transacciones que **commitearon**, para así poder asegurar el valor del item.
Se comienza del **BEGIN CKPT que si tiene su END CKPT**, ya que esto nos asegura que todo lo que esté por encima ya fue escrito en disco. Por lo que, únicamente nos resta rehacer las operaciones por debajo de este, ya que **no podemos saber si fueron o no escritas en el disco**.

Entonces, hago seguimiento REDU de las operaciones commiteadas arrancando luego de la línea 03:

-L04 : hay un **COMMIT T1** , sigo.

-L05 : hay un **BEGIN T2** , sigo.

-L06 : hay un **WRITE de T2** , como esta commiteada, la persisto en el disco.
CAMBIO EN DISCO A→4 (utilizo valor nuevo)

-L07 : hay un **COMMIT T2** , sigo.

-L08 : hay un **END CKPT**, sigo.

-L09, L10, L11, L12, son de T3 la cual no fue commiteada y ya se deshicieron sus cambios previamente.

Finalmente, por cada transacción no commiteada se escribe ABORT en el log y se hace flush del log en el disco. En este caso, se hace con **(ABORT,T3)**.