Integridad de datos en entornos concurrentes

Varias transacciones ejecutándose simultáneamente compartiendo recursos (los datos).

**Mantener la BD consistente** => Se debe evitar los mismos problemas de entorno monousuario (corte de luz, cuelga el servidor,etc).

Entornos concurrentes => **puede quedar la BD de datos inconsistente incluso sin haber fallos.**

**Seriabilidad** =>Garantiza consistencia de la BD.

T0T1 y T1T0 respeta A+B => mantiene consistencia.

T0T1 <> T1T0 => Sin embargo no son iguales, tampoco dejan igual la BD.

Planificación = secuencia de ejecución de transacciones.

* Involucra todas las instrucciones de las transacciones .
* Conservan el orden de ejecución de las mismas.
* Un conjunto de m transacciones generan m! planificaciones en serie.
* La ejecución concurrente no necesita una planificación en serie (ineficiente no aprovechar concurrencia).

Conclusiones:

El programa debe conservar la consistencia.

La inconsistencia temporal puede ser causa de inconsistencia en planificaciones en paralelo, es decir, las instrucciones de las transacciones “chocan”. Esto sucede por las instrucciones READ y WRITE.

Una **planificación concurrente debe equivaler a una planificación en serie.**

**I1, I2 instrucciones de T1 y T2:**

* Si operan sobre datos distintos. No hay conflicto. ✓

Si operan sobre el mismo dato:

* I1 = READ(Q) = I2, no importa el orden de ejecución. ✓
* I1 = READ(Q), I2 = WRITE(Q) depende orden ejecución (I1 leerá valores distintos). ~
* I1 = WRITE(Q), I2 = READ(Q) depende orden ejecución (I2 leerá valores distintos). ~
* I1 = WRITE(Q) = I2, depende el estado final de la BD deseado. ~

Conclusión: I1, I2 está en conflicto si actúan sobre el mismo dato y al menos una es un write.

Una Planificación S se transforma en una S´ mediante intercambios de instrucciones no conflictivas, entonces S y S´ son equivalentes en cuanto a conflictos. Esto significa que si:

S’ es consistente => S también lo será

S’ es inconsistente => S también será inconsistente

**S´ es serializable en conflictos si existe S / sean equivalentes en cuanto a conflictos y S es una planificación serie**.

**Métodos de control de concurrencia:**

Aseguran que la ejecución simultánea de 2 o más transacciones no conflictúa. Se dan estos 2 tipos de control:

* Bloqueo
* Basado en hora de entrada

**Bloqueo:**

1. Compartido Lock\_c(dato)(solo lectura)
2. Exclusivo Lock\_e(dato) (lectura/escritura)

Las transacciones piden lo que necesitan.

Los bloqueos pueden ser compatibles y existir simultáneamente (compartidos)

**Control de concurrencia:** Pido lo que necesito, lo uso y lo libero.

*Se deben llevar los bloqueos de las transacciones al comienzo.*

**Deadlock**: situación en la que una transacción espera un recurso de otra y viceversa.

Cómo lidiar con deadlock?

1. Si los datos se liberan pronto => se evitan posibles deadlock, pero puedo tener inconsistencias.
2. Si los datos se mantienen bloqueados=> hay deadlock, pero se evita inconsistencia.

Se elige la 2da opción, ya que hay que evitar la inconsistencia.

Para solucionar el deadlock lo que se hace es “**elegir a la víctima**”. Consiste en seleccionar una de las 2 transacciones y abortarla, deshaciendo todo lo que haya realizado. De esta manera se liberan los datos que generan deadlock, y así la otra transacción va a poder empezar.

**Protocolo basado en hora de entrada**

* El orden de ejecución se determina por adelantado, no depende de quien llega primero. C/Transacción recibe una HDE que puede ser la hora del servidor o un contador.
* Si HDE(Ti) < HDE(Tj), Ti es anterior.
* C/Dato tiene hora de último WRITE ejecutado y hora de último READ ejecutado.

Algoritmo de ejecución:

Ti Solicita READ(Q)

HDE(Ti) < HW(Q): X (trata de leer un dato que fue escrito por una transacción posterior)

HDE(Ti) HW(Q): O (se ejecuta y se establece HR(Q)=Max{HDE(Ti), HR(Ti)} )

Ti solicita WRITE(Q)

HDE(Ti) < HR(Q): X (Q fue utilizado por otra transacción anteriormente y supuso que no cambiaba)

HDE(Ti) < HW(Q): rechazo (se intenta escribir un valor viejo, obsoleto)

HDE(Ti) > [HW(Q) y HR(Q)]: ejecuta y HW(Q) se establece con HDE(Ti).

Si Ti falla, y se rechaza entonces puede recomenzar con una nueva hora de entrada.