Control de Concurrencia (2)

Grafos de Dependencia

Una forma para determinar si un itinerario no secuencial es consistente o no, es mediante los llamados Grafos de Dependencia.

El grafo de dependencia se define como GD=(N,A), donde:

N: es el conjunto de **nodos** que representan a las transacciones que conforman un itinerario.

A: es el conjunto de **arcos** donde cada arco se establece según se cumpla una o más condiciones, posiblemente, conflictivas entre dos o más transacciones interesadas en el mismo objeto.

Se etiqueta con el nombre del objeto.

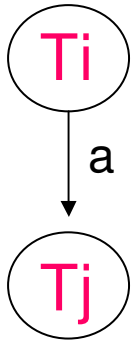
Grafos de Dependencia

1) Ti: write a
   Tj: write a

2) Ti: write a
   Tj: read a

3) Ti: read a
   Tj: write a

Si se da alguna de esas combinaciones entonces se conforma el grafo:



En este grafo, la ejecución de Ti precede a la ejecución de Tj

Grafos de Dependencia

En un itinerario secuencial el grafo de dependencias asociado no contiene ciclos pues si se tiene el arco:

$$Ti \rightarrow Tj$$

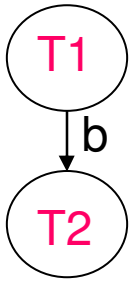
esto indica que Ti precede a Tj en el itinerario.

Se puede decir entonces que para que un itinerario sea serializable, es decir, que aunque no sea secuencial sea equivalente a uno secuencial, basta que el grafo de dependencias asociado no tenga ciclos.

Examinando el ejemplo anterior podemos asociar los siguientes grafos a los itinerarios:

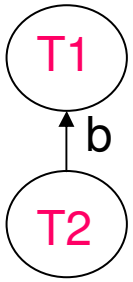
I1	Valor Real	I2	Valor Real	I3	Valor Real
T1: Begin		T1: Begin		T1: Begin	
T1: read a	a=2000	T1: read a	a=2000	T1: read a	a=2000
T1: read b	b=3000	T2: Begin		T2 : Begin	
T1: a=a-1000		T2 : read b	b=3000	T1 : read b	b=3000
T1 : b=b+1000		T2 : read c	c=4000	T2 : read c	c=4000
T1 : write a	a=1000	T1 : a=a-1000		T1 : a=a-1000	
T1 : write b	b=4000	T1 : write a	a=1000	T2 : read b	b =3000
T1 : Commit		T2 : b=b+1000		T1 : b=b+1000	
T2 : Begin		T2 : c=c-1000		T2 : c=c-1000	
T2 : read c	c=4000	T2 : write b	b=4000	T1 : write a	a=1000
T2 : read b	b=4000	T2 : write c	c=3000	T2: b =b+1000	
T2 : c=c-1000		T1 read b	b=4000	T1 : write b	b=4000
T2 : b=b+1000		T2 : Commit		T2 : write c	c=3000
T2 : write c	c=3000	T1 : b=b+1000		T1 : Commit	
T2. write b	b=5000	T1 : write b	b=5000	T2 : write b	b=4000
T2 : Commit		T1 : Commit		T2 : Commit	

Grafo para I1



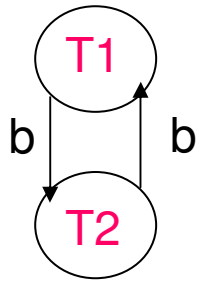
I1	Valor Real	I2	Valor Real	I3	Valor Real
T1: Begin		T1: Begin		T1: Begin	
T1: read a	a=2000	T1: read a	a=2000	T1: read a	a=2000
T1: read b	b=3000	T2: Begin		T2 : Begin	
T1: a=a-1000		T2 : read b	b=3000	T1 : read b	b=3000
T1 : b=b+1000		T2 : read c	c=4000	T2 : read c	c=4000
T1 : write a	a=1000	T1 : a=a-1000		T1 : a=a-1000	
T1 : write b	b=4000	T1 : write a	a=1000	T2 : read b	b =3000
T1 : Commit		T2 : b=b+1000		T1 : b=b+1000	
T2 : Begin		T2 : c=c-1000		T2 : c=c-1000	
T2 : read c	c=4000	T2 : write b	b=4000	T1 : write a	a=1000
T2 : read b	b=4000	T2 : write c	c=3000	T2: b =b+1000	
T2 : c=c-1000		T1 read b	b=4000	T1 : write b	b=4000
T2 : b=b+1000		T2 : Commit		T2 : write c	c=3000
T2 : write c	c=3000	T1 : b=b+1000		T1 : Commit	
T2. write b T2 : Commit	b=5000	T1 : write b T1 : Commit	b=5000	T2 : write b T2 : Commit	b=4000

Grafo para I2



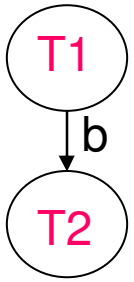
I1	Valor Real	I2	Valor Real	I3	Valor Real
T1: Begin		T1: Begin		T1: Begin	
T1: read a	a=2000	T1: read a	a=2000	T1: read a	a=2000
T1: read b	b=3000	T2: Begin		T2 : Begin	
T1: a=a-1000		T2 : read b	b=3000	T1 : read b	b=3000
T1 : b=b+1000		T2 : read c	c=4000	T2 : read c	c=4000
T1 : write a	a=1000	T1 : a=a-1000		T1 : a=a-1000	
T1 : write b	b=4000	T1 : write a	a=1000	T2 : read b	b =3000
T1 : Commit		T2 : b=b+1000		T1 : b=b+1000	
T2 : Begin		T2 : c=c-1000		T2 : c=c-1000	
T2 : read c	c=4000	T2 : write b	b=4000	T1 : write a	a=1000
T2 : read b	b=4000	T2 : write c	c=3000	T2: b =b+1000	
T2 : c=c-1000		T1 read b	b=4000	T1 : write b	b=4000
T2 : b=b+1000		T2 : Commit		T2 : write c	c=3000
T2 : write c	c=3000	T1 : b=b+1000		T1 : Commit	
T2. write b T2 : Commit	b=5000	T1 : write b T1 : Commit	b=5000	T2 : write b T2 : Commit	b=4000

Grafo para I3

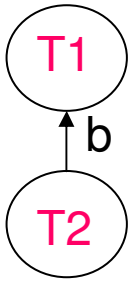


I1	Valor Real	I2	Valor Real	I3	Valor Real
T1: Begin		T1: Begin		T1: Begin	
T1: read a	a=2000	T1: read a	a=2000	T1: read a	a=2000
T1: read b	b=3000	T2: Begin		T2 : Begin	
T1: a=a-1000		T2 : read b	b=3000	T1 : read b	b=3000
T1 : b=b+1000		T2 : read c	c=4000	T2 : read c	c=4000
T1 : write a	a=1000	T1 : a=a-1000		T1 : a=a-1000	
T1 : write b	b=4000	T1 : write a	a=1000	T2 : read b	b =3000
T1 : Commit		T2 : b=b+1000		T1 : b=b+1000	
T2 : Begin		T2 : c=c-1000		T2 : c=c-1000	
T2 : read c	c=4000	T2 : write b	b=4000	T1 : write a	a=1000
T2 : read b	b=4000	T2 : write c	c=3000	T2: b =b+1000	
T2 : c=c-1000		T1 read b	b=4000	T1 : write b	b=4000
T2 : b=b+1000		T2 : Commit		T2 : write c	c=3000
T2 : write c	c=3000	T1 : b=b+1000		T1 : Commit	
T2. write b T2 : Commit	b=5000	T1 : write b T1 : Commit	b=5000	T2 : write b T2 : Commit	b=4000

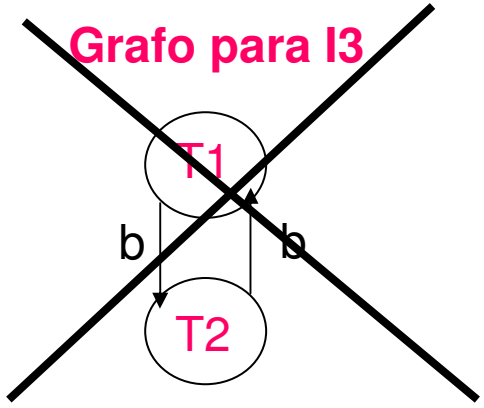
Grafo para I1



Grafo para I2



Grafo para I3



¿Cómo encontrar un itinerario serializable?

MECANISMOS PARA EL CONTROL DE CONCURRENCIA

Con el fin de lograr un itinerario consistente en un SGBD, se establecen una serie de reglas que deben ser verificadas por las transacciones que se realizan concurrentemente en el sistema.

A este conjunto de reglas se le llama: *protocolo*.

En los ***protocolos*** se maneja una idea común: cuando dos transacciones desean a un mismo objeto, se permite que una de ellas lo tome y que la otra pueda accederlo si y sólo si la primera lo libera. A esto se llama *definir cerrojos* sobre los objetos en conflicto.

MECANISMOS PARA EL CONTROL DE CONCURRENCIA

En los protocolos se maneja una idea común: cuando dos transacciones desean a un mismo objeto, se permite que una de ellas lo tome y que la otra pueda accederlo, si y sólo si, la primera lo libera. A esto se llama definir cerrojos sobre los objetos en conflicto.

Las operaciones que permiten definir los cerrojos son:

lock
unlock

En este curso estudiaremos dos protocolos:

1. Exclusión mutua.
2. Dos fases.

MECANISMOS PARA EL CONTROL DE CONCURRENCIA

1. PROTOCOLO DE EXCLUSIÓN MUTUA

Es el protocolo clásico para la administración de recursos. Las reglas de este protocolo se definen de la siguiente manera:

1. Ninguna transacción puede efectuar una actualización o una lectura de un objeto 'a' si no ha adquirido un cerrojo sobre él.

2. Si una transacción no ha podido adquirir un cerrojo sobre un objeto 'a' que está siendo utilizado por otra transacción, entonces la segunda debe esperar hasta que el objeto sea liberado por la primera.

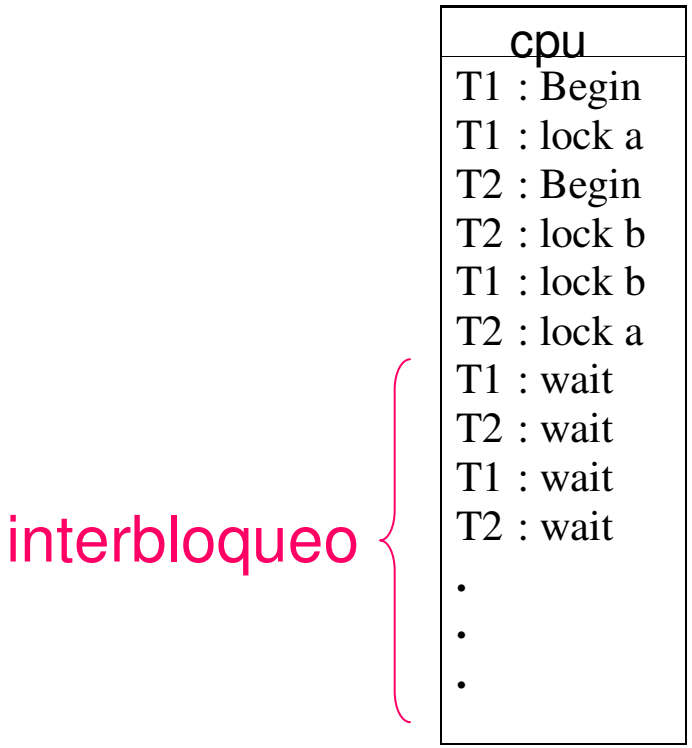
1. PROTOCOLO DE EXCLUSIÓN MUTUA

Ejemplo de la utilización del protocolo de exclusión mutua, utilizando dos transacciones que se realizan concurrentemente y que conforman un itinerario como el siguiente:

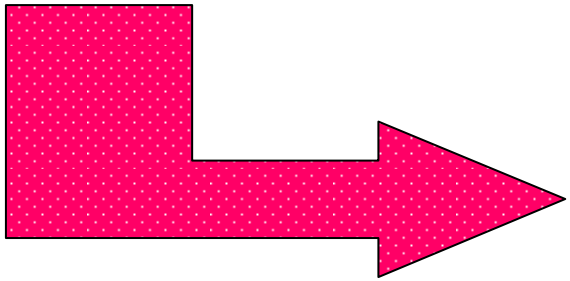
cpu	Valor del Objeto
T1: Begin	
T1: lock a	
T1: read a	a = 1.000
T2: Begin	
T2: lock a	
T1: a= a+3000	a = 4.000
T2: wait	
T1 : write a	
T2 : wait	
T1 : unlock a	
T1 : Commit	
T2 : read a	a = 4.000
T2 : a=a+6000	
T2 : write a	a = 10.000
T2 : unlock a	
T2 : Commit	

1. PROTOCOLO DE EXCLUSIÓN MUTUA

Uno de los problemas que se puede presentar cuando dos o más transacciones se interesan cada una por un objeto que ha sido bloqueado por la otra. Se cae entonces en una espera indefinida por parte de las transacciones involucradas, es llamado abrazo mortal o interbloqueo (*deadlock*, en inglés).



En este ejemplo, T1 ha adquirido un cerrojo sobre el objeto ‘a’. Posteriormente T2 adquiere el control sobre el objeto ‘b’. Luego, T1 se interesa en el objeto ‘b’ y T2 en el objeto ‘a’, sin haber sido liberados por ninguna de las dos transacciones. Así, se entra en una espera indefinida por parte de las dos transacciones.



cpu
T1 : Begin
T1 : lock a
T2 : Begin
T2 : lock b
T1 : lock b
T2 : lock a
T1 : wait
T2 : wait
T1 : wait
T2 : wait
.
.
.

Una forma de resolver esto, es mediante la asignación de un número llamado estampilla de tiempo. Con esto se ordenan las solicitudes de las transacciones.

Esto se implementa o bien con el método sin decomiso (wait die, en inglés) que obliga a una transacción a esperar o a abortar si entra en conflicto con otra transacción cuya estampilla es menor; o también se puede implementar con el método con decomiso (wound wait, en inglés), que es lo contrario al anterior, es decir, si una transacción entra en conflicto con otra que tiene una estampilla menor obliga a ésta a esperar o a abortar, quedándose con el objeto.

MECANISMOS PARA EL CONTROL DE CONCURRENCIA

2. PROTOCOLO DE DOS FASES

Este protocolo refina más el concepto de cerrojo, usando el cerrojo compartido y el cerrojo exclusivo.

CERROJO COMPARTIDO O DE LECTURA.
Se denota s-lock (share lock, en inglés). Si una transacción coloca este cerrojo a un objeto, sólo puede leerlo. Esto permite que el objeto sea compartido, para lectura, por otras transacciones concurrentes.

CERROJO EXCLUSIVO O DE ESCRITURA.
Se denota x-lock (exclusive lock, en inglés). Esto hace que el objeto sea para lectura y escritura sólo de la transacción que le ha puesto el cerrojo. Ninguna otra transacción puede tener acceso a él.

2. PROTOCOLO DE DOS FASES

Una Tabla de Compatibilidad entre Cerrojos sobre un mismo objeto en transacciones concurrentes, sería:

CERROJO	**S-LOCK**	**X-LOCK**
S-LOCK	compatible	conflicto
X-LOCK	conflicto	conflicto

Definición de Transacción bien Formada

Para que una transacción se considere bien formada, debe tener las siguientes propiedades:

- Si la transacción T quiere leer un objeto ‘a’, debe adquirir un cerrojo s-lock a.
- Si la transacción T quiere escribir sobre un objeto ‘a’, debe adquirir un cerrojo x-lock a.
- Después de que una transacción haya llegado a su punto de validación (llegado al Commit) ningún objeto debe quedar bloqueado.

Ejemplo:

Begin transaction T1
s-lock a
read a
a=a-1000
write a
unlock a
x-lock b
read b
b=b+1000
write b
Commit T1

Begin transaction T2
x-lock a
read a
a=a-1000
write a
x-lock b
unlock a
read b
b=b+5000
write b
unlock b
Commit T2

En este caso, T1 no está bien formada puesto que escribió sobre el *objeto a* habiendo adquirido sólo un cerrojo compartido, además, concluyó sin liberar el cerrojo exclusivo que tenía sobre el *objeto b* . Por otro lado, la transacción T2 está bien formada.

2. PROTOCOLO DE DOS FASES

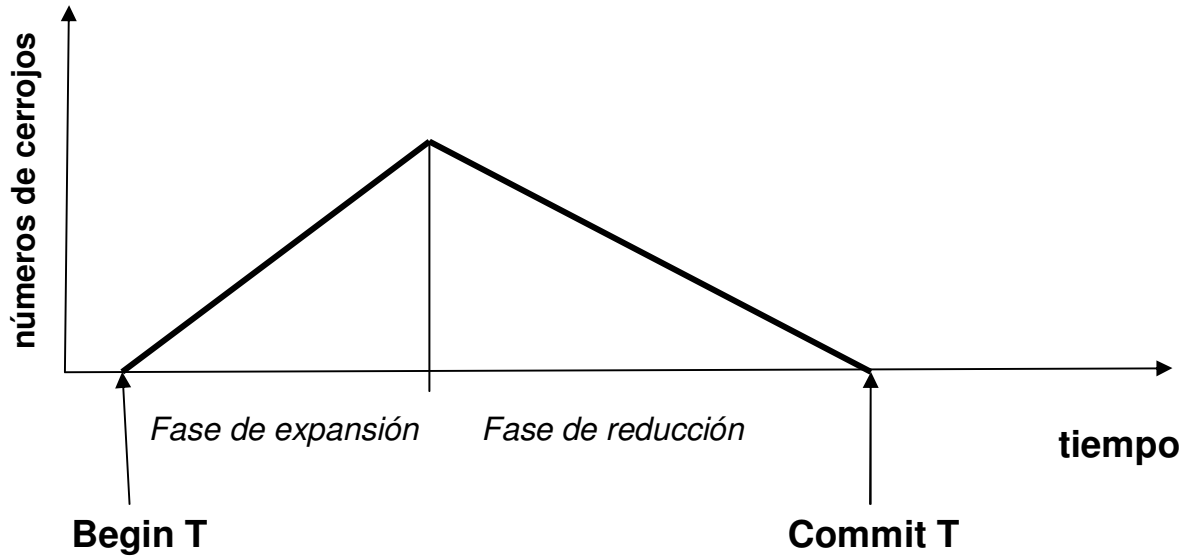
Ahora sí se pueden definir las reglas del protocolo de dos fases:

1. ***Dos transacciones no pueden tener cerrojos en conflicto en forma simultánea.***
2. ***Toda transacción que libera un cerrojo no puede adquirir otro.***

CERROJO	**S-LOCK**	**X-LOCK**
S-LOCK	compatible	conflicto
X-LOCK	conflicto	conflicto

2. PROTOCOLO DE DOS FASES

Las dos fases aluden al hecho de que cuando se adquieren cerrojos hay una etapa de expansión. Por otro lado, al acercarse al *Commit* de la transacción se comienzan a liberar cerrojos, por lo que hay una fase de reducción.



2. PROTOCOLO DE DOS FASES

Resultado importante: si en un itinerario todas las transacciones se encuentran en dos fases, entonces el itinerario es equivalente a un itinerario secuencial.

Ejemplo:

I1	I2
T1 : Begin	T3 : Begin
T1 : x-lock a	T3 : x-lock a
T1 : read a	T3 : read a
T2: Begin	T3: a=a-5000
T2: x-lock b	T3: write a
T2: read b	T3: x-lock b
T1: a=a-1000	T3 : read b
T2: b=b+1000	T3: unlock a
T1: write a	T4 : Begin
T2 : write b	T4 : s-lock a
T1 : unlock a	T3: b=b+5000
T2 : unlock b	T3: write b
T1 : s-lock b	T3: unlock b
T1 : read b	T3 : Commit
T1: b?b+2000	T4 : s-lock b
T1: unlock b	T4 : unlock a
T2 : Commit	T4 : unlock b
T1 : Commit	T4 : Commit

En este caso, se tiene que I1 no satisface el protocolo de dos fases mientras que I2 sí, es consistente I2.

Fuentes consultadas:

[1] Prof. Elsa Liliana Tovar.
Notas de clase compiladas entre 1997-2004.

[2] Ramakrishnan Raghu. ,
“Database Management Systems".

[3] Navathe, Elsamri,
“Fundamentals of DataBase System”.

[4] http://www.slideshare.net/koolkampus/ch15