Base de Datos. 1C2009.
Clase práctica de Transacciones
Docente: Alejandro Eidelsztein

CONTROL DE CONCURRENCIA EN BASES DE DATOS

1. TRANSACCIONES. SERIALIZABILIDAD[1] :

1.1. ITEMS:

Construiremos un modelo para estudiar los problemas de concurrencia en BD.
En este modelo veremos a la BD como un conjunto de *ítems*. Un ítem puede ser un atributo, una tupla o una relación entera. Los denominaremos con letras: A, B, X, Y, etc.

1.2. DEFINICIÓN DE TRANSACCIÓN:

Una *transacción T* es una ejecución de un *programa P* que accede a la BD.
Un mismo programa P puede ejecutarse varias veces. Cada ejecución de P es una transacción Ti.
Una transacción es una sucesión de *acciones* (u operaciones)
Una acción es un paso atómico. Estos pueden ser:

- *leer* un ítem X de Ti: ri[X]
- *escribir* un ítem X de Ti : wi[X],
- *abort* de Ti : ai,
- *commit* de Ti : ci.

$Ti \subseteq \{ri[X], wi[X] / X \text{ es un ítem}\} \cup \{ai, ci\}$

Las Ti se ejecutan en forma concurrente (entrelazada) y esto genera el *problema de interferencia*.

Asimismo, pueden ocurrir fallas en medio de la ejecución y esto genera el *problema de recuperación* que analizaremos en la próxima clase.

Dos problemas clásicos que se pueden presentar son el *lost update* y el *dirty read*.

1.3. LOST UPDATE:

El lost update (actualización perdida) ocurre cuando se pierde la actualización hecha por una transacción T1 por la acción de otra transacción T2 sobre el mismo ítem.

Ejemplo 1:
Supongamos el programa P= Read(A); A:=A+1; Write(A);
y dos ejecuciones de P, T1 y T2 sobre el ítem A, con el siguiente entrelazamiento:

T1	T2	A	< -- (valor del ítem A en disco, inicialmente=5)
Read(A)		5	
	Read(A)	5	
A:=A+1		5	
	A:=A+1	5	
	Write(A)	6	
Write(A)		6	< -- (valor incorrecto de A, se perdió la actualización hecha por T2)

Nota:
Read(A) copia el valor del ítem A en disco a la variable local de la transacción.
Write(A) copia el valor de la variable local de la transacción al ítem A en disco.
A:=A+1 se hace sobre la variable local de la transacción.

[1] En este apunte cuando indiquemos "serializabilidad" (serializability en inglés) nos estaremos refiriendo en todos los casos a "conflict-serializability". De igual forma cuando indiquemos "equivalente" (equivalent) y "serializable" nos estaremos refiriendo a conflict-equivalent y conflict-serializable.

1.4. DIRTY READ:
El dirty read o lectura sucia ocurre cuando una transacción T2 lee un valor de un ítem dejado por otra transacción T1 que abortó luego de que T2 leyera el ítem (T2 se quedó con un valor sucio que será deshecho por el rollback de T1)
Esto además podría producir un fenómeno no deseado como es el *abort en cascada* (si T2 leyó de T1 que abortó, deberíamos abortar T2, luego si T3 leyó de T2 deberíamos abortar a su vez T3 y así sucesivamente)

Ejemplo 2:
Supongamos los programas
P1= Read(A); A:=A-1; Write(A); Read(B); B:=B/A; Write(B); y
P2= Read(A); A:=A*2; Write(A);
y dos ejecuciones: una T1 de P1 sobre los ítems A y B, y una T2 de P2 sobre el ítem A, con el siguiente entrelazamiento:

T1	T2	A	B	< -- (valor de los ítems A y B en disco, inicialmente A=1 y B=2)
Read(A)		1	2	
A:=A-1		1	2	
Write(A)		0	2	
	Read(A)	0	2	
	A:=A*2	0	2	
Read(B)		0	2	
	Write(A)	0	2	
B:=B/A		0	2	< -- (T1 falla por la división por cero y aborta volviendo A al valor anterior, pero T2 ya leyó A)

1.5. PROPIEDADES ACID:
La idea es que dada una BD en un estado consistente, luego de ejecutarse las transacciones la BD quede también en un estado consistente.

Una forma de garantizar esto último es que las transacciones cumplan con las *propiedades ACID*.
Estas propiedades son:
- **A**tomicidad: T se ejecuta completamente o no se ejecuta por completo (todo o nada)
- **C**onsistencia: T transforma un estado consistente de la BD en otro estado consistente (los programas deben ser correctos)
- A**I**slamiento: Las Ti se ejecutan sin interferencias.
- **D**urabilidad: Las actualizaciones a la BD serán durables y públicas.

1.6. HISTORIAS (SCHEDULES)
Por ejemplo, si P= Read(X); X:=X+1; Write(X); Commit;
entonces dos ejecuciones distintas de P, T1 y T2 las escribiremos como:
T1= r1[X] w1[X] c1
T2= r2[X] w2[X] c2

Si T= {T1,T2, ...,Tn} es un conjunto de transacciones, entonces una *historia* (o *schedule*) H sobre T es:
H= Ui=1,n Ti
Donde H respeta el orden de las acciones de cada Ti.

Ejemplo 3:
Si T1= r1[X w1[X] c1 y T3= r3[X] w3[Y] w3[X] c3, una historia H1 sobre el conjunto de transacciones {T1,T3} y el conjunto de ítems {X,Y} podría ser:

H1= r1[X] r3[X] w1[X] c1 w3[Y] w3[X] c3

También podemos expresar H1 en forma tabular:

T1	T3
r[X]	
	r[X]
w[X]	
c	
	w[Y]
	w[X]
	c

1.7. EQUIVALENCIA DE HISTORIAS:
Dos historias H y H' son *equivalentes* (H≡H') si:
1. Si están definidas sobre el mismo conjunto de transacciones.
2. Las operaciones *conflictivas* tienen el mismo orden.

Dos operaciones de Ti y Tj (i≠j) son *conflictivas* si operan sobre el mismo ítem y al menos alguna de las dos es un write.

1.8. HISTORIAS SERIALES:
H es *serial* (Hs) si para todo par de transacciones Ti y Tj en H, todas las operaciones de Ti preceden a las de Tj o viceversa.
Las historias seriales (y las equivalentes a estas) son las que consideraremos como correctas.

1.9. HISTORIAS SERIALIZABLES:
H es *serializable* (SR) si es equivalente a una historia serial (Hs)

1.10. GRAFO DE PRECEDENCIA:
Dado H sobre T= {T1,T2, ...,Tn}, un SG para H, *SG(H)*, es un grafo dirigido cuyos nodos son los Ti y cuyos arcos Ti → Tj (i≠j), tal que alguna operación de Ti precede y conflictua con alguna operación de Tj en H.

1.11. CONSTRUCCIÓN DEL GRAFO DE PRECEDENCIA (MODELO READ/WRITE):
Algoritmo:
1. Hacer un nodo por cada Ti y
2. Si alguna operación de Ti precede y conflictúa con alguna operación de Tj en H (i<>j), luego hacer un arco Ti → Tj

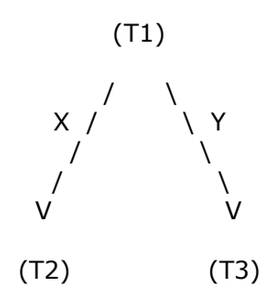
1.12. TEOREMA 1 DE SERIALIZABILIDAD:
H es SR si y solo si SG(H) es acíclico.

H es equivalente a cualquier Hs serial que sea un *ordenamiento topológico* de SG(H)

Ejemplo 4:
Dados :
T1= w[X] w[Y] c
T2= r[X] w[X] c
T3= r[Y] w[Y] c

H= w1[X] w1[Y] c1 r2[X] r3[Y] w2[X] c2 w3[Y] c3

SG(H):



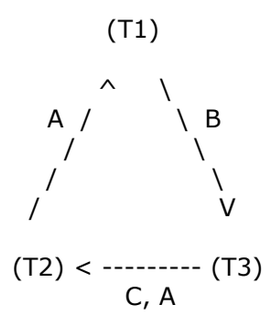
Vemos que H es SR (serializable) y es equivalente a las historias seriales:

H´= T1 T2 T3
H” = T1 T3 T3

Ejemplo 5:
Dados:
T1= r[A] w[B]
T2= r[C] w[A]
T3= r[A] w[C] w[B]

H= r3[A] w3[C] r2[C] w2[A] r1[A] w1[B] w3[B]

SG(H):



Vemos que SG(H) tiene un ciclo, por lo tanto H no es SR.

2. LOCKING:

2.1. DEFINICIÓN DE LOCK:
El *lock* es un privilegio de acceso a ítem de la BD.

Decimos que una Ti setea u obtiene un lock sobre el item X.

Al usar locking aparecen dos problemas que consideraremos en la próxima clase: *Livelocks* y *Deadlocks.*

2.2. LOCKING BINARIO (Exclusive locks):
Este modelo de locking tiene 2 estados o valores:

Locked:	Lock(X)	li[X]
Unlocked:	Unlock(X)	ui[X]

El lock binario fuerza exclusión mutua sobre un ítem X.

Ejemplo:
Si reescribimos el programa P que producía lost update como:
P= Lock(A); Read(A); A:=A+1; Write(A); Unlock(A);
y hacemos la historia H con T1 y T2 veremos que el lost update no se produce.

2.3. LOCKING TERNARIO (Shared/Exclusive locks):
Este modelo permite mayor concurrencia que el binario.
Tiene 3 estados o valores:

Read locked:	RLock(X)	rli[X]	(lock compartido)
Write locked:	WLock(X)	wli[X]	(lock exclusivo)
Unlocked:	ULock(X)	uli[X] o ui[X]	

2.4. MODELO SIMPLIFICADO BASADO EN LOCKING:
En este modelo una transacción T es vista como una secuencia de locks y unlocks (Hacemos abstracción de las otras operaciones)

2.5. MATRIZ DE COMPATIBILIDAD DE LOCKING (CONFLICTOS):

		Lock sostenido por Tj:	
		RLOCK	**WLOCK**
Lock pedido	**RLOCK**	Y	N
por Ti:	**WLOCK**	N	N

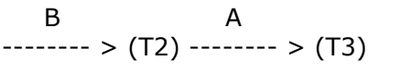
2.6. REGLAS DE LEGALIDAD DE LOCKING:
H es legal si:
1. Una Ti no puede leer ni escribir un ítem X hasta tanto no haya hecho un lock de X.
2. Una Ti que desea obtener un lock sobre X que ha sido lockeado por Tj en un modo que conflictúa, debe esperar hasta que Tj haga unlock de X.

2.7. CONSTRUCCIÓN DEL GRAFO DE PRECEDENCIA (MODELO LOCKING BINARIO: LOCK/UNLOCK):
Algoritmo:
1. Hacer un nodo por cada Ti
2. Si Ti hace Unlock de X y luego Tj hace Lock de X (i<>j), luego hacer un arco Ti -> Tj

Ejemplo 6:
Dada:
H= l2[A] u2[A] l3[A] u3[A] l1[B] u1[B] l2[B] u2[B]

Vemos que H es legal

Para ver si es SR hacemos el SG(H):



H es SR y es equivalente a T1 T2 T3.

2.8. CONSTRUCCIÓN DEL GRAFO DE PRECEDENCIA (MODELO LOCKING TERNARIO: RLOCK/WLOCK/UNLOCK):

Algoritmo:

1. Hacer un nodo por cada Ti
2. Si Ti hace RLock o WLock de X, y luego Tj es la próxima que hace WLock de X (i<>j), hacer un arco Ti -> Tj
3. Si Ti hace WLock de X y Tj (i<>j) hace RLock de X, (antes que cualquier otra haga WLock de X), luego hacer un arco Ti -> Tj

Ejemplo 7:
Dados:
T1= rl[A] wl[B] ul[A] ul[B]
T2= rl[A] ul[A] rl[B] ul[B]
T3= wl[A] ul[A] wl[B] ul[B]
T4= rl[B] ul[B] wl[A] ul[A]

H= wl3[A] rl4[B] ul3[A] rl1[A] ul4[B] wl3[B] rl2[A] ul3[B] wl1[B] ul2[A] ul1[A] wl4[A] ul1[B] rl2[B] ul4[A] ul2[B]

Vemos que H es legal
Si hacemos el SG(H) veremos que tiene ciclos y por lo tanto no es SR.

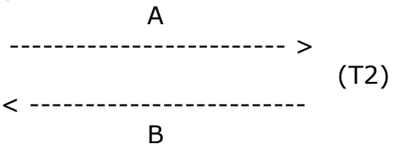
2.9. LOCKING Y SERIALIZABILIDAD:

Ahora nos podríamos preguntar si al usar locking (y H es legal) obtendremos siempre historias serializables. Veamos un contraejemplo.

Ejemplo 8:
T1= l[A] u[A] l[B] u[B]
T2= l[A] l[B] u[A] u[B]

H= l1[A] u1[A] l2[A] l2[B] u2[A] u2[B] l1[B] u1[B]

SG(H):



Vemos que H es legal pero no es serializable.

Observamos que el mecanismo de locking por si solo no garantiza serializabilidad. Se necesita agregar un *protocolo* para posicionar los locks y unlocks.
La idea es usar un protocolo de dos fases en cada transacción. Una primera fase de crecimiento donde la transacción va tomando todos los ítems (locks) y luego una segunda fase de decrecimiento donde los va liberando (unlocks)

2.10. PROTOCOLO 2PL (Two Phase Locking):

T es 2PL si todos los locks preceden al primer unlock.

2.11. TEOREMA 2 DE SERIALIZABILIDAD:

Dado T={T1, T2, …, Tn}, si toda Ti en T es 2PL, entonces todo H sobre T es SR.

Ejemplo 9:
Si volvemos a considerar el Ejemplo 7 donde H no es SR veremos que T2, T3 y T4 no son 2PL.

Nota del docente:

Algunos de los ejemplos de este apunte fueron tomados de los siguientes libros:

"Concurrency Control and Recovery in Database Systems", de Philip A. Bernstein, Vassos Hadzilacos y Nathan Goodman, 1987.

"Introducción a las Bases de Datos Relacionales", de Alberto Mendelzon y Juan Ale, 2000.

"Principles of Database and Knowledge-Base Systems", Volume I, de Jeffrey Ullman, 1988.