Contenido:

7. Transacciones y control de la concurrencia

Los SGBDs son sistemas concurrentes, i.e., admiten la ejecución concurrente de consultas.
Ejemplo: Sistema de venta de billetes de avión.
Por tanto, es necesario:
Modelo de procesos concurrentes para admitir operaciones concurrentes que preserven la integridad de los datos.

7.1. Conceptos básicos

7.1.1. Transacción

Una transacción es una unidad de la ejecución de un programa. Puede consistir en varias operaciones de acceso a la base de datos. Está delimitada por constructoras como begin-transaction y end-transaction.

7.1.2. Propiedades ACID

7.1.2.1. Atomicidad

Es la propiedad de las transacciones que permite observarlas como operaciones atómicas: ocurren totalmente o no ocurren.
Casos a considerar:
- Consultas unitarias. Incluso para consultas unitarias hay que preservar la atomicidad: en un sistema operativo de tiempo compartido, la ejecución concurrente de dos consultas SQL puede ser incorrecta si no se toman las precauciones adecuadas.
- Operación abortada. Por ejemplo, debido a una división por cero; por privilegios de acceso; o para evitar bloqueos

7.1.2.2. Consistencia

La ejecución aislada de la transacción conserva la consistencia de la base de datos.

7.1.2.3. Aislamiento

Para cada par de transacciones que puedan ejecutarse concurrentemente Ti y Tj, se cumple que para los efectos de Ti:
- Tj ha terminado antes de que comience Ti
- Tj ha comenzado después de que termine Ti
Las transacciones son independientes entre sí.

7.1.2.3.1. Niveles de aislamiento

Se puede ajustar el nivel de aislamiento entre las transacciones y determinar para una transacción el grado de aceptación de datos inconsistentes.
A mayor grado de aislamiento, mayor precisión, pero a costa de menor concurrencia.
El nivel de aislamiento para una sesión SQL establece el comportamiento de los bloqueos para las instrucciones SQL.
Niveles de aislamiento:

- Lectura no comprometida. Menor nivel. Asegura que no se lean datos corruptos físicamente.
- Lectura comprometida. Sólo se permiten lecturas de datos comprometidos.
- Lectura repetible. Las lecturas repetidas de la misma fila para la misma transacción dan los mismos resultados.
- Secuenciable. Mayor nivel de aislamiento. Las transacciones se aíslan completamente.

Comportamiento concurrente de las transacciones.

- Lectura sucia. Lectura de datos no comprometidos. (Retrocesos)
- Lectura no repetible. Se obtienen resultados inconsistentes en lecturas repetidas.
- Lectura fantasma. Una lectura de una fila que no existía cuando se inició la transacción.

	Comportamiento permitido		
Nivel de aislamiento	**Lectura sucia**	**Lectura no repetible**	**Lectura fantasma**
Lectura no comprometida	Sí	Sí	Sí
Lectura comprometida	No	Sí	Sí
Lectura repetible	No	No	Sí
Secuenciable	No	No	No

SQL Server permite todos estos niveles, Oracle sólo permite la lectura comprometida y secuenciable. Los niveles se pueden establecer en ambos SGBD para cada transacción.

7.1.2.4. Durabilidad

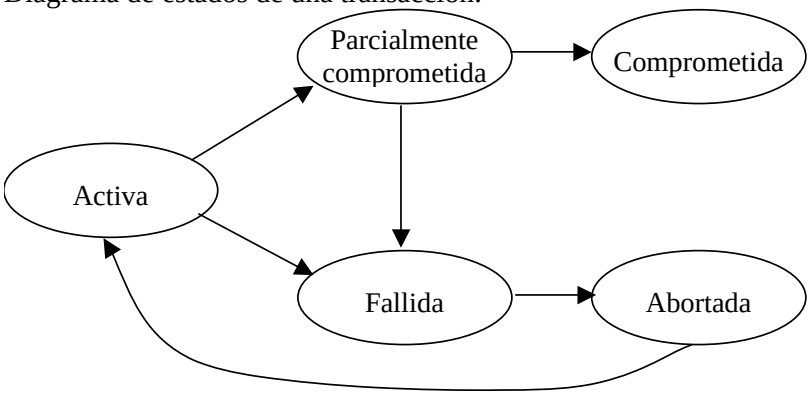
El sistema gestor de bases de datos asegura que perduren los cambios realizados por una transacción que termina con éxito.

7.1.3. Estados de una transacción

- Activa: Durante su ejecución

- Parcialmente comprometida: Después de ejecutar su última instrucción.
- Fallida: Imposible de continuar su ejecución normal.
- Abortada: Transacción retrocedida y base de datos restaurada al estado anterior a su ejecución. Se puede reiniciar o cancelar.

Diagrama de estados de una transacción:



Retroceso de transacciones: Automáticos y programados.

7.1.4. Programación de transacciones

Tipos de transacciones: implícitas (modo de autocompromiso) y explícitas (delimitadas).
Transacciones explícitas:

Oracle	SQL Server
	BEGIN TRAN[SACTION]
Instrucción 1	Instrucción 1
...	...
SAVEPOINT *sp*	SAVE TRAN[SACTION] *sp*
...	...
ROLLBACK [TO SAVEPOINT *sp*]	ROLLBACK [TRAN[SACTION] *sp*]
...	...
Instrucción n	Instrucción n
COMMIT [WORK]	COMMIT [TRAN[SACTION]]

Ejemplo (SQL Server):
Incremento de un 1% de las comisiones 15% y 16% de la tabla de comisiones *roysched*. Si no existen estos porcentajes entonces no se ejecutará la instrucción de actualización. En este ejemplo se deben incrementar ambos; si uno de ellos no existe, se debe dejar sin modificar.

```
BEGIN TRAN actualiza_comisiones  -- Inicio de la transacción
  USE pubs
IF EXISTS (SELECT titles.title, roysched.royalty
             FROM titles, roysched
             WHERE titles.title_id=roysched.title_id
                   AND roysched.royalty=16)
      UPDATE roysched SET royalty=17 WHERE royalty=16
  ELSE
    ROLLBACK TRAN actualiza_comisiones

IF EXISTS (SELECT titles.title, roysched.royalty
             FROM titles, roysched
             WHERE titles.title_id=roysched.title_id
                   AND roysched.royalty=15)
    BEGIN
      UPDATE roysched SET royalty=16 WHERE royalty=15
      COMMIT TRAN actualiza_comisiones
    END
  ELSE
    ROLLBACK TRAN actualiza_comisiones
```

7.1.4.1. Transacciones anidadas

Una transacción anidada o multinivel T consiste en un conjunto $T = \{t_1, t_2, . . ., t_n\}$ de subtransacciones y en un orden parcial P sobre T.
Cada t_i de T puede abortar sin obligar a que T aborte. Puede que T reinicie t_i o simplemente no ejecute t_i. Si se compromete t_i, esa acción no hace que t_i sea permanente, sino que t_i *se compromete con* T, y puede que aborte si T aborta. La ejecución de T no debe violar el orden parcial P. Es decir, si aparece un arco $t_i \rightarrow t_j$ en el grafo de precedencia, $t_j \rightarrow t_i$ no debe estar en el cierre transitivo de P.

Ejemplo (SQL Server):

```
USE MyDB
GO
CREATE PROCEDURE Formular_pedido AS --Crea un procedimiento almacenado
  BEGIN TRAN Tran_formular_pedido
  -- Instrucciones SQL para la formulación del pedido
  COMMIT TRAN Tran_formular_pedido

GO

BEGIN TRAN Tran_pedidos
  -- Formular un pedido
  EXEC Formular_pedido
COMMIT TRAN Tran_pedidos

GO
```

Oracle no admite transacciones anidadas, SQL Server sí.

7.1.5. Elementos

Los elementos son las unidades de datos para los que se controla el acceso.
Por ejemplo: relación, tupla, campos, bloques, ...
La granularidad es el tamaño de los elementos. Así, se habla de sistemas de grano fino o de grano grueso, para denotar elementos pequeños o grandes, respectivamente.
A mayor granularidad, menor concurrencia.
No obstante, para determinadas operaciones es interesante bloquear relaciones enteras, como con la reunión de relaciones (join).

7.1.6. Bloqueos

Un bloqueo es una información del tipo de acceso que se permite a un elemento. El SGBD impone los bloqueos necesarios en cada momento. El gestor de acceso a los datos implementa las restricciones de acceso. En algunos sistemas se permite que el usuario pueda indicar el bloqueo más adecuado (locking hints).
Tipos de bloqueo con respecto a la operación:
read-locks: sólo permite lectura
write-locks: permite lectura y escritura
El gestor de bloqueos almacena los bloqueos en una tabla de bloqueos:
(<elemento>, <tipo de bloqueo>, <transacción>)=(E,B,T)
La transacción T tiene un tipo de bloqueo B sobre el elemento E.
Normalmente, E es clave, aunque no siempre, porque varias transacciones pueden bloquear el mismo elemento de forma diferente.

Niveles de bloqueo
Especifica la granularidad del bloqueo

- Fila: Fila individual

- Clave: Fila de un índice
- Página: Páginas (8KB)
- Extent: Extensión (grupo de 8 páginas contiguas de datos o índices)
- Table: Tabla completa
- Database: Base de datos completa

Modos de bloqueo

Especifica el modo en que se bloquea un elemento

- Compartido: para operaciones sólo de lectura. Se permiten lecturas concurrentes, pero ninguna actualización.
- Actualización: para operaciones que *pueden* escribir. Sólo se permite que una transacción adquiera este bloqueo. Si la transacción modifica datos, se convierte en exclusivo, en caso contrario en compartido.
- Exclusivo. para operaciones que *escriben* datos. Sólo se permite que una transacción adquiera este bloqueo.
- Intención: se usan para establecer una jerarquía de bloqueo. Por ejemplo, si una transacción necesita bloqueo exclusivo y varias transacciones tienen bloqueo de intención, no se concede el exclusivo.
  - Intención compartido. Bloqueo compartido.
  - Intención exclusivo. Bloqueo exclusivo.
  - Compartido con intención exclusivo. Algunos bloqueos compartidos y otros exclusivos.
- Esquema. para operaciones del DDL.
- Actualización masiva. En operaciones de actualización masiva

Control de concurrencia

P: READ A; A:=A+1; WRITE A;

A en la BD	5	5	5	5	6	6
T1	READ A		A:=A+1			WRITE A
T2		READ A		A:=A+1	WRITE A	
A en T1	5	5	6	6	6	6
A en T2		5	5	6	6	

P: LOCK A; READ A; A:=A+1; WRITE A; UNLOCK A;

A en la BD		5	5	6	6	6	6	7	7
T1	LOCK A	READ A	A:=A+1	WRITE A	UNLOCK A				
T2			LOCK A	LOCK A	LOCK A	READ A	A:=A+1	WRITE A	UNLOCK A
A en T1		5	6	6	6				
A en T2						6	7	7	7

Livelock

Espera indefinida de una transacción por un bloqueo que no se llega a conceder porque se cede a otras transacciones.

Una solución (sistemas operativos): estrategia first-come-first-served (se atiende al primero que llega).

Deadlock

T1: LOCK A; LOCK B; UNLOCK A; UNLOCK B;

T2: LOCK B; LOCK A; UNLOCK B; UNLOCK A;

T1 y T2 bloquean A y B => Espera indefinida de T1 y T2.

Soluciones (sistemas operativos):

1- Concesión simultánea de *todos* los bloqueos de una transacción.

2- Asignar un orden lineal arbitrario a los elementos y requerir que las transacciones pidan los bloqueos en este orden.

3- Permitir los deadlocks y analizar cada cierto tiempo si existen.

7.1.7. Secuencialidad de planificaciones

La ejecución concurrente de varias transacciones es correcta <=> su efecto es el mismo si se ejecutan secuencialmente en cualquier orden.
Una *planificación* para un conjunto de transacciones es el orden en el que se realizan los pasos elementales de las transacciones.
Una planificación es *secuencial* si todos los pasos de cada transacción ocurren consecutivamente.
Una planificación es *secuenciable* si su efecto es equivalente al de la planificación secuencial.
Ej: Se transfieren 10 unidades de A a B y 20 de B a C.
T1: READ A; A:=A-10; WRITE A; READ B; B:=B+10;WRITE B;
T2: READ B; B:=B-20; WRITE B; READ C; C=C+20; WRITE C;
Cualquier planificación secuencial preserva A+B+C (Sólo hay dos, T1 antes de T2 o viceversa)

T1	T2	T1	T2	T1	T2
READ A		READ A		READ A	
A:=A-10			READ B	A:=A-10	
WRITE A		A:=A-10			READ B
READ B			B:=B-20	WRITE A	
B:=B+10		WRITE A			B:=B-20
WRITE B			WRITE B	READ B	
	READ B	READ B			WRITE B
	B:=B-20		READ C	B:=B+10	
	WRITE B	B:=B+10			READ C
	READ C		C=C+20	WRITE B	
	C=C+20	WRITE B			C=C+20
	WRITE C		WRITE C		WRITE C
Planificación secuencial		Planificación secuenciable pero no secuencial		Planificación no secuenciable (B se incrementa en 10 unidades en lugar de decrementarse; problema que se resuelve con bloqueos)	

7.1.8. Planificadores y protocolos

Un *planificador* arbitra los conflictos de acceso. Resuelve los livelocks con first-come-first-served. Puede también manejar deadlocks y no secuencialidad con:
- Espera de transacciones por liberación de bloqueos
- Abortos y reinicios de transacciones.

La espera produce en general más bloqueos pendientes.
Los bloqueos pendientes provocan deadlocks.
Los deadlocks se resuelven generalmente abortando al menos una transacción.
Un *protocolo* es una restricción sobre la secuencia de pasos atómicos de una transacción.
Por ejemplo, el orden impuesto para prevenir deadlocks.

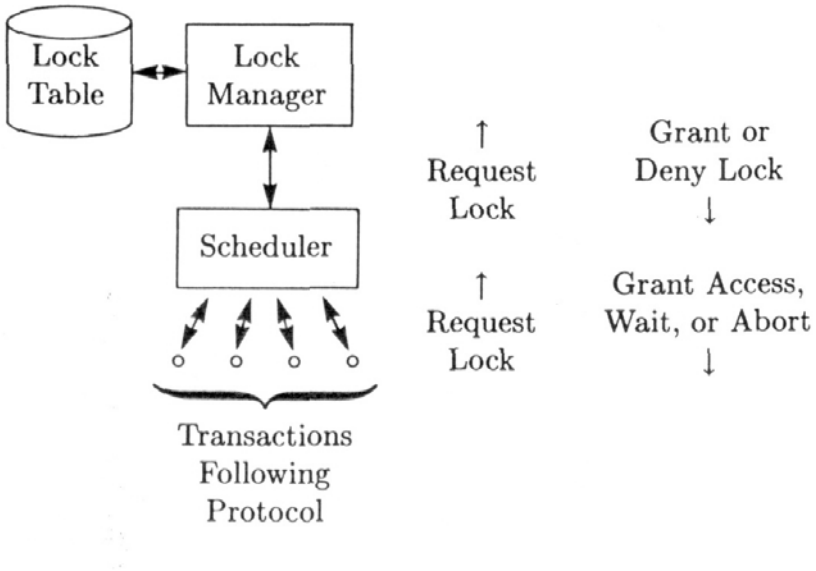


Figure 9.3 Protocol, scheduler, and lock manager.

7.2. Modelo transaccional simple

Una transacción es una secuencia de instrucciones LOCK y UNLOCK.
LOCK asume una lectura de un elemento y UNLOCK una escritura.
La secuencialidad en este modelo simple implica secuencialidad en modelos más complejos.

7.2.1. Semántica de transacciones

Es el significado de la ejecución de la transacción (qué hace).
Para diseñar protocolos y planificadores es necesario relacionar la semántica (informal) de las transacciones con un test que determine si una secuencia de pasos de transacciones entrelazadas es secuenciable.
Primero: ver que la semántica es apropiada (conservadora: puede prohibir planificaciones que sean secuenciables pero no permite las que no lo sean).
Segundo: se hace corresponder la semántica con un grafo de ejecución que permite decidir si una planificación es secuenciable.
Se asocia una función f al par LOCK A y UNLOCK A. Argumentos: todos los elementos bloqueados anteriormente a UNLOCK A. A_0 es el valor inicial de A.
Los *valores* de A son fórmulas que se construyen aplicando estas funciones a los valores iniciales.
Se asume que fórmulas diferentes tienen valores diferentes.
Dos planificaciones son *equivalentes* si sus fórmulas para el valor final de cada elemento son iguales.

T1		T2		T3	
LOCK A		LOCK B		LOCK A	
LOCK B		LOCK C		LOCK C	
UNLOCK A	$f_1(A,B)$	UNLOCK B	$f_3(B,C)$	UNLOCK C	$f_6(A,C)$
UNLOCK B	$f_2(A,B)$	LOCK A		UNLOCK A	$f_7(A,C)$
		UNLOCK C	$f_4(A,B,C)$		
		UNLOCK A	$f_5(A,B,C)$		

	Step	A	B	C
(1)	T_1: LOCK A	A_0	B_0	C_0
(2)	T_2: LOCK B	A_0	B_0	C_0
(3)	T_2: LOCK C	A_0	B_0	C_0
(4)	T_2: UNLOCK B	A_0	$f_3(B_0, C_0)$	C_0
(5)	T_1: LOCK B	A_0	$f_3(B_0, C_0)$	C_0
(6)	T_1: UNLOCK A	$f_1\big(A_0, f_3(B_0, C_0)\big)$	$f_3(B_0, C_0)$	C_0
(7)	T_2: LOCK A	$f_1\big(A_0, f_3(B_0, C_0)\big)$	$f_3(B_0, C_0)$	C_0
(8)	T_2: UNLOCK C	$f_1\big(A_0, f_3(B_0, C_0)\big)$	$f_3(B_0, C_0)$	(i)
(9)	T_2: UNLOCK A	(ii)	$f_3(B_0, C_0)$	(i)
(10)	T_3: LOCK A	(ii)	$f_3(B_0, C_0)$	(i)
(11)	T_3: LOCK C	(ii)	$f_3(B_0, C_0)$	(i)
(12)	T_1: UNLOCK B	(ii)	$f_2\big(A_0, f_3(B_0, C_0)\big)$	(i)
(13)	T_3: UNLOCK C	(ii)	$f_2\big(A_0, f_3(B_0, C_0)\big)$	(iii)
(14)	T_3: UNLOCK A	(iv)	$f_2\big(A_0, f_3(B_0, C_0)\big)$	(iii)

Key:

(i): $f_4\Big(f_1\big(A_0, f_3(B_0, C_0)\big), B_0, C_0\Big)$ $\quad$ (iii): $f_6\big((ii),\ (i)\big)$

(ii): $f_5\Big(f_1\big(A_0, f_3(B_0, C_0)\big), B_0, C_0\Big)$ $\quad$ (iv): $f_7\big((ii),\ (i)\big)$

Figure 9.5 A schedule.

La planificación no es secuenciable:

Si T1 precede a T2 en la planificación secuencial, el valor final de B es $f_3(f_2(A_0, B_0), C_0)$ en lugar de $f_2(A_0, f_3(B_0, C_0))$.

Si T2 precede a T1, el valor final de A aplicaría f_1 a una expresión con f_5. En la figura anterior esto no sucede, por tanto, T2 no puede preceder a T1 en una planificación secuencial equivalente.

T1 no puede preceder a T2 ni T2 puede preceder a T1 en una planificación secuencial equivalente => no existe dicha planificación secuencial equivalente.

7.2.2. Test de secuencialidad

Para determinar si un planificador es correcto, se demuestra que cada planificación que admite es secuenciable.

Se examina la planificación con respecto al orden en que se bloquean elementos. Este orden debe ser consistente con la planificación secuencial equivalente. Si hay dos secuencias con órdenes diferentes de transacciones, estos dos órdenes no son consistentes con una planificación secuencial.

El problema se reduce a la búsqueda de ciclos en un grafo dirigido.

Algoritmo:

Entrada: Una planificación S para las transacciones T1,..,Tk.

Salida: Determina si S es secuenciable y, si lo es, produce una planificación secuencial equivalente a S.

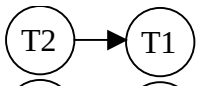
Método:

Creación de un grafo de secuencialización G cuyos nodos son transacciones.

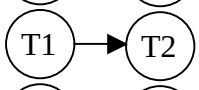
S=a1;a2;..;an

ai es Tj: LOCK Am o Tj: UNLOCK Am
Dado ai= Tj: UNLOCK Am se busca ap= Ts: LOCK Am en el orden de S tal que s<>j. Entonces, el arco <Tj,Ts> pertenece a G.
Intuitivamente: en cualquier planificación secuencial equivalente a S, Tj debe preceder a Ts.
Si G tiene un ciclo, S no es secuenciable.
Si G no tiene ciclos, se produce una secuencia de transacciones mediante la ordenación topológica (también determina si no hay ciclos):
1. Encontrar un nodo Ti sin arcos de entrada (si no se encuentra, hay un ciclo).
2. Listar Ti y eliminarlo.
3. Si quedan nodos, ir a 1.
Ej. del caso anterior:

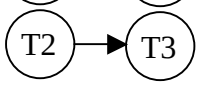
(4) T2: UNLOCK B
(5) T1: LOCK B



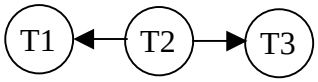
(6) T1: UNLOCK A
(7) T2: LOCK A



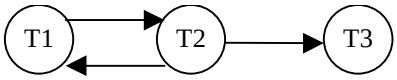
(8) T2: UNLOCK C
(11) T3: LOCK C



(9) T2: UNLOCK A
(1) T1: LOCK A
(10) T3: LOCK A



En definitiva:



Teorema 6.1: El algoritmo determina correctamente si una planificación es secuenciable.
Demostración: Ejercicio.

7.3. Protocolo de bloqueo de dos fases

Requisito: todos los bloqueos preceden a los desbloqueos. Primera fase: bloqueos. Segunda: desbloqueos.
Propiedad: según este requisito no existen planificaciones no secuenciables legales.
Teorema 9.2: Si S es cualquier planificación de transacciones de dos fases, S es secuenciable.
Demostración: Supongamos que no sea secuenciable. Entonces, por el teorema 9.1, el grafo de secuencialización de G para S tiene un ciclo:

$$T_{i1} \rightarrow T_{i2} \rightarrow \cdots \rightarrow T_{ip} \rightarrow T_{i1}$$

Por tanto, algún bloqueo de T_{i2} sigue a un desbloqueo de T_{i1}; algún bloqueo de T_{i3} sigue a un desbloqueo de T_{i2}, ..., algún bloqueo de T_{i1} sigue a un desbloqueo de T_{ip}. Por tanto, algún bloqueo de T_{i1} sigue a un desbloqueo de T_{i1}, contradiciendo la suposición de que T_{i1} es una transacción de dos fases.
El protocolo de dos fases no asegura ausencia de interbloqueos:
T1= LOCK B; LOCK A; UNLOCK A; UNLOCK B;
T2= LOCK A; LOCK B; UNLOCK B; UNLOCK A;

T1	T2
LOCK B	
	LOCK A
	LOCK B
LOCK A	
¡Interbloqueo!	

7.4. Protocolos basados en grafos

A menudo es útil observar el conjunto de elementos de datos de la base de datos como una estructura de grafo. Por ejemplo:

1. Organización lógica o física de los elementos.
2. Definición de elementos de varios tamaños, donde los grandes engloban a los pequeños. Ej: relacional: tupla $\subset$ bloque $\subset$ relación $\subset$ base de datos.
3. Control de concurrencia efectivo.

Se pueden diseñar protocolos que no sean de dos fases pero que aseguren la secuencialidad.

En general, sea $D = \{d_1, d_2, \ldots, d_n\}$ el conjunto de todos los elementos de datos de la base de datos dotado de un orden parcial $\rightarrow$. Si en el grafo existe un arco $d_i \rightarrow d_j$, entonces la transacción que acceda tanto a d_i como a d_j debe acceder primero a d_i y después a d_j.

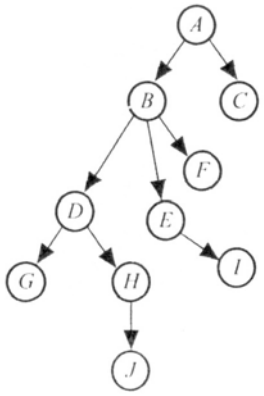
7.4.1. Protocolo de árbol

Caso particular de protocolo basado en grafos, grafos que sean árboles con raíz.

Reglas:

1. Cada transacción T_i bloquea al menos un elemento.
2. El primer bloqueo de T_i puede ser sobre cualquier elemento.
3. Sucesivos bloqueos de T_i sólo pueden ser sobre elementos cuyo padre haya bloqueado T_i.
4. Los elementos se pueden desbloquear en cualquier momento.
5. T_i no puede bloquear de nuevo un elemento que haya bloqueado y desbloqueado anteriormente.

Ej:



T_1: LOCK *B*; LOCK *E*; LOCK *D*; UNLOCK *B*; UNLOCK *E*; LOCK *G*; UNLOCK *D*; UNLOCK *G*;
T_2: LOCK *D*; LOCK *H*; UNLOCK *D*; UNLOCK *H*;
T_3: LOCK *B*; LOCK *E*; UNLOCK *E*; UNLOCK *B*;
T_4: LOCK *D*; LOCK *H*; UNLOCK *D*; UNLOCK *H*;

Planificación secuenciable:

T_1	T_2	T_3	T_4
LOCK *B*			
	LOCK *D* LOCK *H* UNLOCK *D*		
LOCK *E* LOCK *D* UNLOCK *B*			

UNLOCK *E*			
		LOCK *B* LOCK *E*	
	UNLOCK *H*		
LOCK *G* UNLOCK *D*			
			LOCK *D* LOCK *H* UNLOCK *D* UNLOCK *H*
		UNLOCK *E* UNLOCK *B*	
UNLOCK *G*			

Se puede demostrar que el protocolo de árbol no sólo asegura la secuencialidad en cuanto a conflictos, sino que también asegura la ausencia de interbloqueos.

El protocolo de bloqueo de árbol tiene la ventaja sobre el protocolo de bloqueo de dos fases de que los desbloqueos se pueden dar antes. El hecho de desbloquear antes puede llevar a unos tiempos de espera menores y a un aumento de la concurrencia. Adicionalmente, debido a que el protocolo está libre de interbloqueos, no se necesitan retrocesos.

Sin embargo, el protocolo tiene el inconveniente de que, en algunos casos, una transacción puede que tenga que bloquear elementos de datos a los que no accede. Por ejemplo, una transacción que tenga que acceder a los elementos de datos A y J en el ejemplo anterior, debe bloquear no sólo A y J sino también los elementos de datos B, D y H. Estos bloqueos adicionales producen un aumento del coste de los bloqueos, la posibilidad de tiempos de espera adicionales y un descenso potencial de la concurrencia.

Además, sin un conocimiento previo de los elementos de datos que es necesario bloquear, las transacciones tienen que bloquear la raíz del árbol y esto puede reducir considerablemente la concurrencia.

Pueden existir planificaciones secuenciables que no se pueden obtener por medio del protocolo de árbol.

Hay planificaciones que son posibles por medio del protocolo de bloqueo de dos fases que no son posibles por medio del protocolo de árbol y viceversa.

Algoritmo para construir un orden secuencial de las transacciones:

1. Crear un nodo por cada transacción.
2. Sean T_i y T_j transacciones que bloquean el mismo elemento, y FIRST(T) el primer elemento bloqueado por T.
3. Si FIRST(T_i) es independiente de FIRST(T_j) (no son "parientes"=pertenecen a árboles disjuntos), el protocolo garantiza que T_i y T_j no bloquearán un nodo en común, por lo que no se traza un arco entre ellas.
4. Si FIRST(T_i) es antepasado de FIRST(T_j) entonces si T_i bloquea FIRST(T_j) antes que T_j, se dibuja un arco $T_i \rightarrow T_j$; si sucede al contrario se dibuja un arco $T_j \rightarrow T_i$.

Se puede demostrar que el grafo resultante es acíclico y que cualquier ordenación topológica del grafo es un orden secuencial para las transacciones. [Ullman]

7.5. Protocolos basados en marcas temporales

Se usan cuando no se imponen bloqueos pero se sigue asegurando secuencialidad.

7.5.1. Marcas temporales

Transacciones:

Cada T_i lleva asociada una marca temporal fijada $MT(T_i)$.

Si T_i se selecciona antes que T_j, entonces $MT(T_i) < MT(T_j)$.

El valor de $MT(T_i)$ puede extraerse del reloj del sistema o con contadores lógicos de transacciones.

Elementos:

Cada elemento de datos D lleva asociado dos marcas temporales:
MTR(D): mayor marca temporal de todas las transacciones que ejecutan con éxito READ D;
MTW(D): mayor marca temporal de todas las transacciones que ejecutan con éxito WRITE D;

7.5.2. Protocolo de ordenación por marcas temporales

Asegura que todas las operaciones leer y escribir conflictivas se ejecutan en el orden de las marcas temporales.

1. Supóngase que la transacción T_i ejecuta READ(D).
   a. Si MT(T_i) < MTW(D) entonces T_i necesita leer un valor de D que ya se ha sobrescrito. Por tanto se rechaza la operación READ y T_i se retrocede.
   b. Si MT(T_i) ≥ MTW(D) entonces se ejecuta la operación READ y MTR(D) se asigna al máximo de MTR(D) y de MT(T_i).
2. Supóngase que la transacción T_i ejecuta WRITE(D).
   a. Si MT(T_i) < MTR(D) entonces el valor de D que produce T_i se necesita previamente y el sistema asume que dicho valor no se puede producir nunca. Por tanto, se rechaza la operación WRITE y T_i se retrocede.
   b. Si MT(T_i) < MTW(D) entonces T_i está intentando escribir un valor de D obsoleto. Por tanto, se rechaza la operación WRITE y T_i se retrocede.
   c. En otro caso se ejecuta la operación WRITE y MT(T_i) se asigna a MTW(D).

Ejemplo:

T_1: READ B;
READ A;
PRINT $A + B$.

La transacción T_2 transfiere 10.000 pts. de la cuenta A a la B y muestra después el contenido de ambas:

T_2: READ B;
$B := B - 10.000$;
WRITE B;
READ A;
$A := A + 10.000$;
WRITE A;
PRINT $A + B$.

T_1	T_2
READ B	
	READ B $B := B - 10.000$ WRITE B
READ A	
	READ A
PRINT $A + B$	
	$A := A + 10.000$ WRITE A PRINT $A + B$

El protocolo de ordenación por marcas temporales asegura la secuencialidad. Esta afirmación se deduce del hecho de que las operaciones conflictivas se procesan durante la ordenación de las marcas temporales.
El protocolo asegura la ausencia de interbloqueos, ya que ninguna transacción tiene que esperar.

7.6. Protocolos optimistas

Cuando los conflictos entre las transacciones son raros se pueden aplicar técnicas optimistas, evitando los protocolos anteriores (más costosos). Estas técnicas asumen que no habrá conflictos en las planificaciones y evitan los costosos bloqueos. Cuando se intente comprometer una transacción se determinará si ha existido conflicto o no. En su caso, la transacción se retrocederá y volverá a iniciarse.
Si los conflictos son raros también lo serán los retrocesos, con lo que el nivel de concurrencia aumentará, aunque también hay que tener en cuenta el coste de los posibles retrocesos en términos de tiempo (todas las actualizaciones se hacen en memoria, no hay retrocesos en cascada porque los cambios no se hacen en la base de datos).
Fases de un protocolo de control de concurrencia optimista:

- Fase de lectura. Desde el inicio de la transacción hasta justo antes de su compromiso. Las actualizaciones se realizan en memoria, no en la propia base de datos.
- Fase de validación. Es la siguiente a la de lectura. Se comprueba que no se viola la secuencialidad. Casos:
  - Transacciones sólo de lectura. Se reduce a comprobar que los valores actuales en la base de datos son los mismos que se leyeron inicialmente.
  - Transacciones con actualizaciones. Se determina si la transacción deja a la base de datos en un estado consistente.

  En ambos casos, si la secuencialidad no se conserva, la transacción se retrocede y se reinicia.
- Fase de escritura. Es posterior a la fase de validación cuando se mantiene la secuencialidad. Consiste en escribir en la base de datos los cambios producidos en la copia local.

Fase de validación:
Cada transacción T recibe una marca temporal (*Inicio(T)*) al iniciar su ejecución, otra al iniciar su fase de validación (*Validación(T)*) y otra al terminar (*Fin(T)*).
Para pasar el test de validación debe ser cierto uno de:
1) Para todas las transacciones S con marcas temporales anteriores a T => *Fin(S) < Inicio(T).*
2) Si la transacción T empieza antes de que acabe otra S anterior, entonces:
   a) Los elementos escritos por S no son los leídos por T, y
   b) La transacción S completa su fase de escritura antes de que T comience su fase de validación, es decir, *Inicio(T) < Fin(S) < Validación(T)*

7.7. Gestión de fallos de transacciones

Causas de aborto:
1. Fallo de la transacción: interrupción por el usuario, fallo aritmético, privilegios de acceso...
2. Deadlock->aborto de una transacción
3. Algoritmos de secuencialidad.
4. Error software o hardware

Fácil: 1, 2 y 3. Difícil: 4. Puntos de recuperación por copias de seguridad.

7.7.1. Compromiso de transacciones

Transacciones activas. En ejecución
Transacciones completadas. Sólo pueden abortar por causa grave: 4.
Punto de compromiso: COMMIT. Momento a partir del cual se entienden completadas.
Las transacciones comprometidas ni se retroceden ni se rehacen.

7.7.2. Datos inseguros

	T1	T2
1	LOCK A	
2	READ A	

3	A:=A-1	
4	WRITE A	
5	LOCK B	
6	UNLOCK A	
7		LOCK A
8		READ A
9		A:=A*2
10	READ B	
11		WRITE A
12		COMMIT
13		UNLOCK A
14	B:=B/A	
15	*¡Fallo!* (división por 0)-> abortar T1	

Acciones después del fallo de T1:
1. UNLOCK B
2. UNDO (4) (de un registro que se verá posteriormente)
3. ROLLBACK T2, porque maneja valores de A inseguros
4. ROLLBACK Ti, para todo Ti que haya usado el dato A inseguro a partir de 14: Retroceso (rollback) en cascada.

7.7.3. Bloqueo de dos fases estricto

Se usa para solucionar el problema anterior.
1. Una transacción no puede escribir en la base de datos hasta que se haya alcanzado su punto de compromiso. (Evita los retrocesos en cascada)
2. Una transacción no puede liberar ningún bloqueo hasta que haya finalizado de escribir en la base de datos, i.e., los bloqueos no se liberan hasta después del punto de compromiso.

7.8. Recuperación de caídas

Tipos de caídas:
- Error de memoria volátil.
- Error de memoria permanente.

Problema: asegurar la atomicidad de las escrituras de las transacciones. Puede haber una caída del sistema antes de que se hayan escrito todos los datos modificados por una transacción.

7.8.1. Recuperación basada en el registro (log)

Es la técnica más habitual.
Almacena consecutivamente los cambios de la base de datos y el estado de cada transacción.
Las tuplas de cuatro elementos significan: (Transacción, Elemento, Valor nuevo, Valor anterior)

Acción	**Entrada del registro**
begin-transaction	(T,begin)
T: WRITE A (A=v)	(T,A,v,A_0)
T: COMMIT	(T, commit)
Abortar T	(T, abort)

Ejemplo anterior:

Paso	**Entrada del registro**
Antes de 1	(T1,begin)
4	(T1,A,9,10)
Antes de 7	(T2,begin)

11	(T2,A,18,9)
12	(T2, commit)
Después de 14	(T1, abort)

Es fundamental que el registro de escritura se cree antes de modificar la base de datos.
Generalmente el registro histórico se implementa en almacenamiento estable.
Hay dos técnicas principales de implementar este tipo de recuperación: modificación diferida e inmediata de la base de datos.

7.8.1.1. Modificación diferida de la base de datos

Pasos:

- Todas las operaciones de escritura se anotan en el registro histórico.
- Cuando la transacción está comprometida parcialmente (se han realizado todas sus operaciones pero aún no se ha modificado la base de datos) se llevan a cabo las escrituras pendientes examinando el registro histórico.
- A continuación se puede anotar la transacción como comprometida.

Si ocurre una caída entre las escrituras, la transacción se puede deshacer.

7.8.1.2. Modificación inmediata de la base de datos

Pasos:

- Todas las operaciones de escritura se anotan en el registro histórico.
- Después de cada operación de escritura se realiza la escritura (modificaciones no comprometidas).
- Finalmente, se anota la transacción como comprometida.

Si ocurre una caída entre las escrituras, la transacción se puede deshacer.

7.8.2. Fallos de memoria permanente

Soluciones:
1) Salvados periódicos de la instalación.
2) Salvados periódicos de la base de datos.

7.9. Bibliografía

- ULLMAN, J.D. "Principles of Databases and Knowledge Base Systems", Vol. I, Computer Science Press, 1998
- SILBERSCHATZ, A., KORTH, H.F., SUDARSHAN, S. "Fundamentos de bases de datos", 3ª edición, McGraw-Hill, 1998.
- ATZENI, P., STEFANO, C., PARABOSCHI, S., TORLONE, R., “Database Systems. Concepts, Languages and Architectures”, McGraw-Hill, 2000.
- CONNOLLY, T., BEGG, C., STRACHAN, A., "Database Systems. A Practical Approach to Design, Implementation, and Management", 2nd edition, Addison-Wesley,1998.
- GARCIA, M.F., REDING , J., WHALEN, E.,DeLUCA, S.A., "Microsoft SQL Server 2000 Administrator's Companion", Microsoft, 2000.