Transacciones

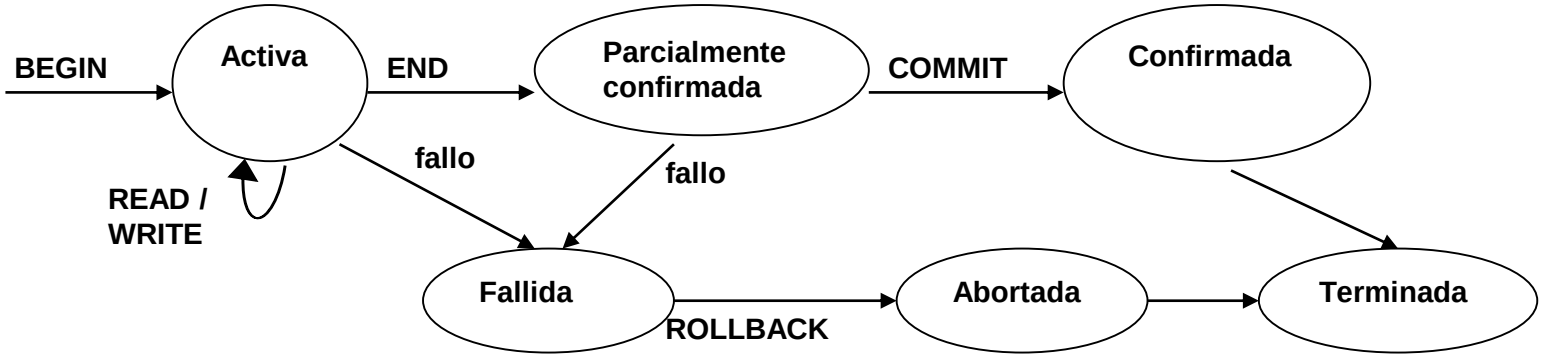
Control de Concurrencia y Recuperación

Base de Datos II

Concepto de transacción

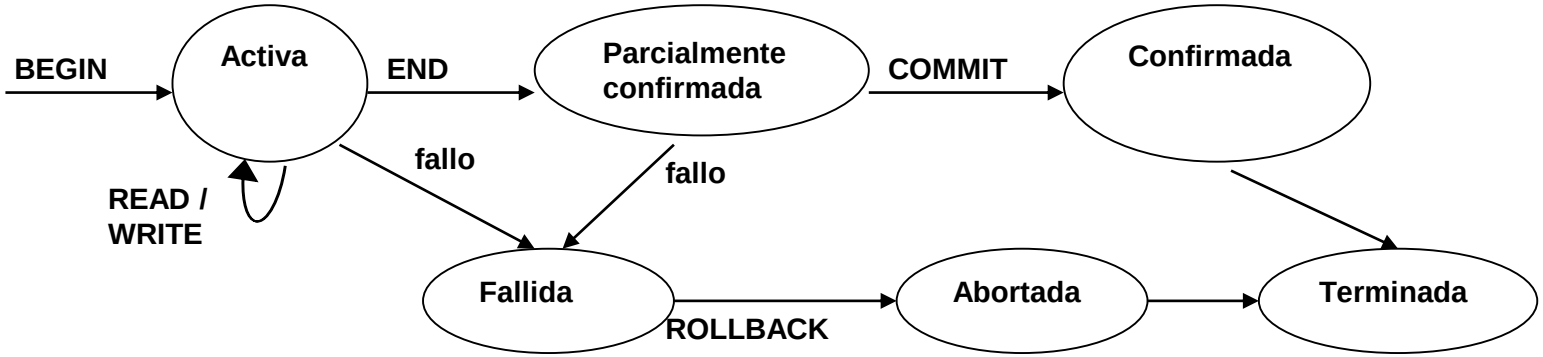
- Una transacción es un conjunto de operaciones, que acceden y posiblemente actualizan los datos, y que son tratados como una unidad.
- Las operaciones de la transacción pueden estar insertas en:
  - un programa externo,
  - un procedimiento almacenado,
  - o ser parte de una secuencia de instrucciones SQL.
- Las transacciones no pueden violar ninguna restricción de integridad de la BD, es decir, la BD debe continuar consistente luego que la transacción termine con éxito.
- Luego de que una transacción ha concluido exitosamente (**confirmado**) los cambios realizados por la misma deben ser permanentes aun ante fallos.

Estados de una transacción



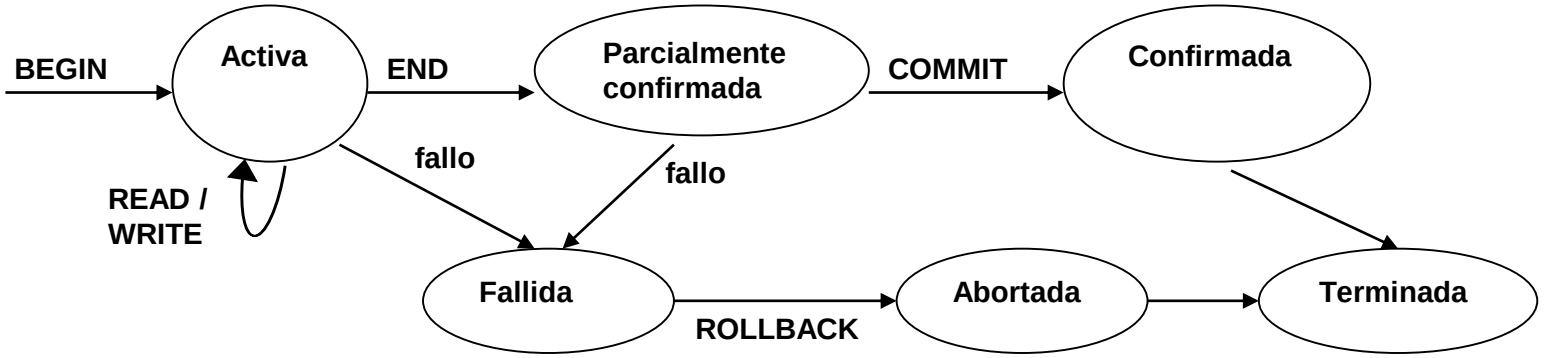
- Una transacción es una unidad atómica de trabajo que es realizada completamente o no realiza ninguna parte de ella.
- El conjunto de operaciones que forman una transacción constituye una unidad lógica de proceso.

Estados de una transacción



- Las operaciones son las siguientes:
  - **BEGIN**: Marca el comienzo de la ejecución de la transacción.
  - **READ** ó **WRITE**: Especifican operaciones de lectura y/o escrituras de ítems que son ejecutadas como parte de una transacción.
  - **END**: Indica que las operaciones de lectura y escritura de la transacción han concluido y marca el fin de la ejecución de la transacción.
  - **COMMIT**: Señala el final adecuado de la transacción y cualquier modificación puede ser transferida definitivamente a la BD y no será deshecha.
  - **ROLLBACK**: Indica que la transacción ha concluido de forma inadecuada y por tanto, cualquier cambio hecho a la BD debe ser deshecho.

Estados de una transacción



Las operaciones hacen que una transacción cambie de un estado a otro. Los estados son los siguientes:

- **Activa**: es el estado al que pasa luego de comenzar su ejecución, y no cambia de estado durante las operaciones de lectura y escritura de ítems.
- **Parcialmente** confirmada: pasa a este estado cuando acaba (END). En este punto se realizan controles de concurrencia, y en el caso de fallo, los cambios no se graban de forma permanente.
- **Confirmada**: punto de confirmación y conclusión de la ejecución.
- **Fallida**: si uno de los chequeos falla o si es abortada durante su estado activo.
- **Abortada**: se deshacen los efectos de sus operaciones de escritura sobre la BD.
- **Terminada**: pasa a este estado cuando la transacción desaparece del sistema.

Equivalencia de operaciones en Oracle

- Una transacción comienza cuando se inicia una sesión.
- Termina al comprometer (commit) o abortar (rollback) la transacción.
- Implícitamente hace un commit al ejecutar una instrucción DDL.

Operación	**Comportamiento**
Commit	Confirma la serie de actualizaciones realizadas desde la apertura de la sesión o desde el último commit.
Rollback	Deshace la serie de actualizaciones desde la apertura de la sesión o desde el último commit.
Savepoint <nomb>	Crea un punto de recuperación dentro de una transacción.
Rollback to savepoint <nomb>	Deshace la serie de actualizaciones realizadas desde el punto de recuperación especificado en <nomb>.

Propiedades de las transacciones (ACID)

Para preservar la integridad de los datos las siguientes propiedades de las transacciones deben ser aseguradas por el SGBD:

- **Atomicidad (Atomicity)**: Una transacción es una unidad atómica de procesamiento; es realizada enteramente o no es realizada en nada.
- **Consistencia (Consistency)**: La ejecución de la transacción debe pasar la BD desde un estado consistente a otro también consistente.
- **Aislamiento (Isolation)**: Una transacción no deberá hacer visible sus modificaciones a otras transacciones hasta que esté confirmada.
- **Persistencia (Durability)**: Cuando una transacción cambia la BD y los cambios son confirmados, estos cambios no deben perderse por fallos posteriores.

Transacciones concurrentes

- La concurrencia se da cuando varias transacciones se ejecutan a la vez.
- La concurrencia permite:
  - Mayor Productividad y Mejor Utilización de los Recursos.
  - Reducción del Tiempo de Respuesta.
- Si se deja sin control la ejecución concurrente de transacciones es posible que deje a la BD en estado inconsistente.
- El parte del SGBD que realiza el control de concurrencia se denomina ***componente de control de concurrencia***.
  - Se encarga de asegurar que el efecto de la ejecución concurrente de un conjunto de transacciones sea igual a que las transacciones se hubiesen ejecutado secuencialmente (lo cual siempre es correcto).

Transacciones concurrentes

Las únicas operaciones significativas para el control de concurrencia y recuperación son:

- Leer_item(X): Lee el ítem X y almacena su valor en memoria.
- Escribir_item(X): Graba el valor de la variable de memoria del ítem X a disco.

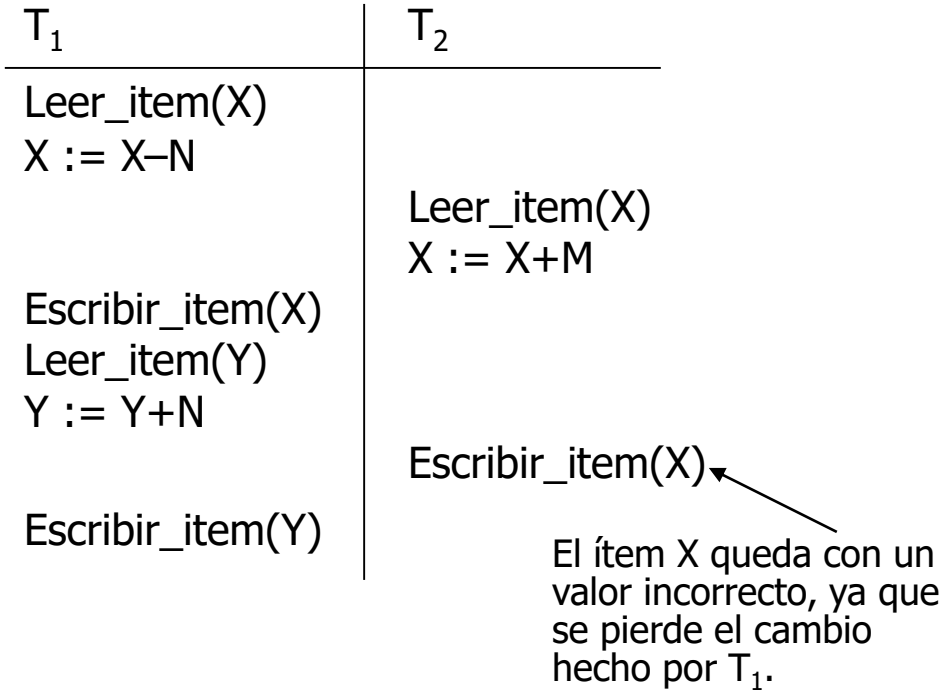
Pasos	Leer X	Escribir X
1. Encontrar la dirección del bloque de disco que contiene los datos del ítem X.	x	x
2. Copiar el bloque de disco a un buffer en memoria principal.	x	x
3. Copiar el ítem X desde el buffer a la variable X.	x	
4. Copiar el ítem X desde la variable X a su localización correcta en el buffer.		x
5. Almacenar el buffer modificado en el bloque de disco.		x

Los problemas de concurrencia

Existen varias formas en las que una transacción puede producir una respuesta incorrecta si alguna otra interfiere de alguna forma.

El problema de la actualización perdida

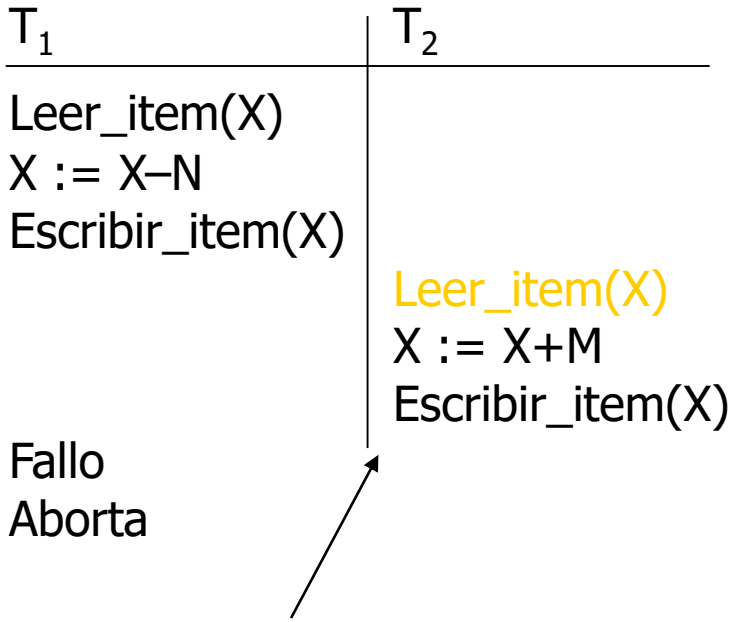
Ocurre cuando dos o más transacciones acceden a los mismos datos y tienen sus operaciones intercaladas de forma que en algún momento queda un valor incorrecto en algún dato al perderse una actualización.



Los problemas de concurrencia

El problema de la lectura sucia

Ocurre cuando una transacción actualiza la BD y luego falla por alguna razón. El ítem modificado es accedido por otra transacción antes de que el cambio sea deshecho y el ítem vuelva a su valor original.

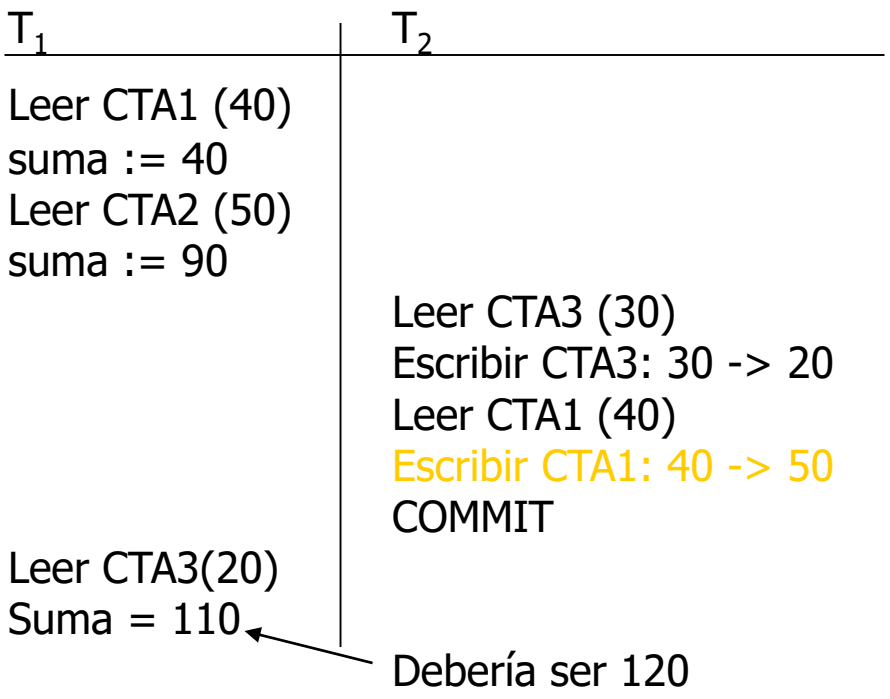


La transacción T_1 falla y debe cambiar el valor de X a su viejo valor, pero T_2 ha leído el valor temporal incorrecto.

Los problemas de concurrencia

El problema del análisis inconsistente

Ocurre cuando una transacción realiza operaciones sobre algunos datos y otra transacción modifica los valores de los mismos.



El problema de la lectura no repetible

Ocurre cuando una transacción T1 lee un ítem de datos dos veces y otra transacción T2 modifica dicho ítem entre las dos lecturas. Por tanto T1 recibe dos valores diferentes del mismo ítem en sus lecturas.

Planificación de transacciones

- Una **planificación** para un conjunto de transacciones consiste en un ordenamiento de todas las operaciones de dichas transacciones, pero conservando el orden interno en cada transacción individual.
- Para dos transacciones T1 y T2, sin el intercalado de sus operaciones sólo son posibles dos ordenamientos:
  - Ejecutar todas las operaciones de T_1 seguidas por todas las operaciones de T_2.
  - Ejecutar todas las operaciones de T_2 seguidas por todas las operaciones de T_1.
  - **Planificaciones Secuenciales**, para **n** transacciones existe **n!** planificaciones diferentes

T_1	T_2
Leer_item(X)	
X := X–N	
Escribir_item(X)	
Leer_item(Y)	
Y := Y+N	
Escribir_item(Y)	
	Leer_item(X)
	X := X+M
	Escribir_item(X)

Plan A

T_1	T_2
	Leer_item(X)
	X := X+M
	Escribir_item(X)
Leer_item(X)	
X := X–N	
Escribir_item(X)	
Leer_item(Y)	
Y := Y+N	
Escribir_item(Y)	

Plan B

Secuencialidad de los planes

- La ejecución concurrente de varias transacciones es correcta si y sólo si su efecto es el mismo que el obtenido al realizar las mismas transacciones en forma secuencial.
- **Un plan secuencial siempre es correcto.**
- Un plan *S* de ***n*** transacciones es **secuenciable** si es equivalente a algún plan secuencial de las mismas ***n*** transacciones.
- Decir que un plan no secuencial *S* es secuenciable es equivalente a decir que es correcto.
- Existen dos formas de definir la equivalencia de planes:
  - Equivalencia por conflictos
  - Equivalencia en cuanto a vistas

Secuencialidad en cuanto a conflictos

Considérese una planificación P en la cual hay dos operaciones I_1 e I_2 pertenecientes a las transacciones T_1 y T_2.

- Si I_1 e I_2 se refieren a distintos ítems de datos, las operaciones se pueden intercambiar sin afectar al resultado del plan.
- Si I_1 e I_2 se refieren al mismo ítem de dato X, entonces el orden de las operaciones puede ser importante.

Operaciones	**Importancia del orden de ejecución**
I_1 = leer_item(X) I_2 = leer_item(X)	El orden no importa, porque leen el mismo valor.
I_1 = leer_item(X) I_2 = escribir_item(X)	El orden es importante y debe mantenerse: - Si I_1 esta antes, no lee el valor de X que escribe I_2 de T_2. - Si I_2 está antes, entonces T_1 lee el valor de X escrito por T_2.
I_1 = escribir_item(X) I_2 = leer_item(X)	El orden es importante y debe mantenerse por la misma razón que el caso anterior.
I_1 = escribir_item(X) I_2 = escribir_item(X)	El orden importa y debe mantenerse porque el valor que se guarda en la base de datos es el último.

Secuencialidad en cuanto a conflictos

Se dice que dos transacciones tienen conflicto si realizan operaciones sobre un mismo dato, y una de sus operaciones es **escribir**.

T_1	T_2
Leer_item(X)	
X := X–N	
Escribir_item(X)	
	Leer_item(X)
	X := X+M
	Escribir_item(X)
Leer_item(Y)	
Y := Y+N	
Escribir_item(Y)	

Plan D

- La operación escribir_item(X) de T1 está en conflicto con la operación leer_item(X) de T2 porque operan sobre el mismo ítem.
- La operación escribir_item(X) de T2 no está en conflicto con la operación leer_item(Y) porque operan sobre distintos datos.
- Se puede cambiar el orden de escribir_item(X) y leer_item(Y) para obtener una nueva planificación P' equivalente a P.

Secuencialidad en cuanto a conflictos

Se dice que dos transacciones tienen conflicto si realizan operaciones sobre un mismo dato, y una de sus operaciones es **escribir**.

T_1	T_2
Leer_item(X)	
X := X–N	
Escribir_item(X)	
	Leer_item(X)
	X := X+M
Leer_item(Y)	
	Escribir_item(X)
Y := Y+N	
Escribir_item(Y)	

Plan D

- La operación escribir_item(X) de T1 está en conflicto con la operación leer_item(X) de T2 porque operan sobre el mismo ítem.
- La operación escribir_item(X) de T2 no está en conflicto con la operación leer_item(Y) porque operan sobre distintos datos.
- Se puede cambiar el orden de escribir_item(X) y leer_item(Y) para obtener una nueva planificación P', el cual debe ser equivalente a P.

Secuencialidad en cuanto a conflictos

Se dice que dos transacciones tienen conflicto si realizan operaciones sobre un mismo dato, y una de sus operaciones es **escribir**.

T_1	T_2
Leer_item(X)	
X := X–N	
Escribir_item(X)	
Leer_item(Y)	
	Leer_item(X)
	X := X+M
	Escribir_item(X)
Y := Y+N	
Escribir_item(Y)	

Plan D

- La operación escribir_item(X) de T1 está en conflicto con la operación leer_item(X) de T2 porque operan sobre el mismo ítem.
- La operación escribir_item(X) de T2 no está en conflicto con la operación leer_item(Y) porque operan sobre distintos datos.
- Se puede cambiar el orden de escribir_item(X) y leer_item(Y) para obtener una nueva planificación P', el cual debe ser equivalente a P.

Secuencialidad en cuanto a conflictos

Se dice que dos transacciones tienen conflicto si realizan operaciones sobre un mismo dato, y una de sus operaciones es **escribir**.

T_1	T_2
Leer_item(X)	
X := X–N	
Escribir_item(X)	
Leer_item(Y)	
Y := Y+N	
Escribir_item(Y)	
	Leer_item(X)
	X := X+M
	Escribir_item(X)

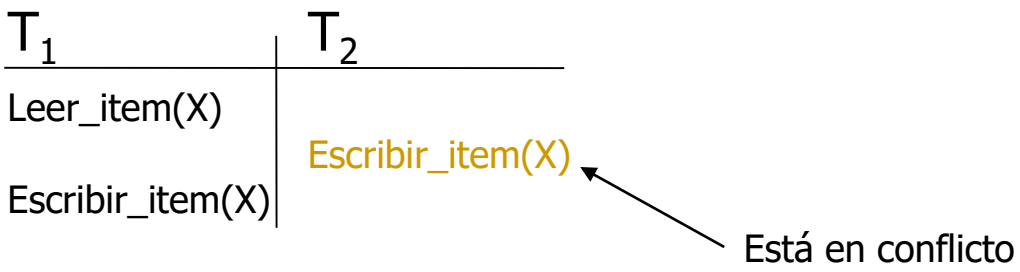
Plan D

- Si una planificación P puede transformarse en otra P' por una serie de intercambios de instrucciones, se dice que P y P' son ***equivalentes en cuanto a conflictos***.
- Si una planificación P puede convertirse en una planificación secuencial P', se dice que P es ***secuenciable en cuanto a conflictos***

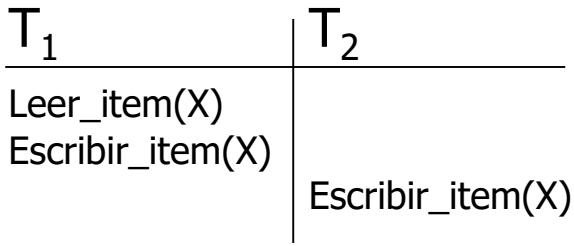
Secuencialidad en cuanto a conflictos

Se dice que una planificación P es **secuenciable en cuanto a conflictos** si es equivalente a una planificación secuencial.

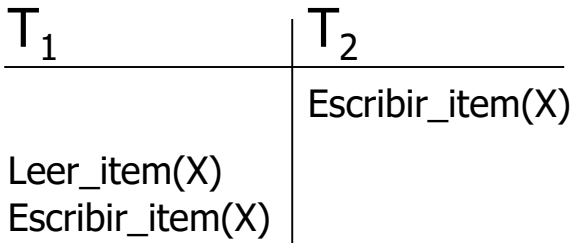
Plan E



Plan secuencial T_1,T_2



Plan secuencial T_2,T_1



Secuencialidad en cuanto a vistas

Dos planes S_1 y S_2, en los cuales participan las mismas transacciones T_1, T_2, ..., T_n , son equivalentes en cuanto a vistas si:

1. Por cada ítem X, si la transacción T_i ha leído el valor inicial de X en S_1, entonces T_i debe leer el valor inicial de X en S_2
2. Por cada ítem X, si la transacción T_i realiza la operación Leer_item(X) en el plan S_1, y el valor lo ha producido la transacción T_j, entonces en el plan S_2 la transacción T_i también debe leer el valor de X que haya producido la transacción T_j.
3. Para cada ítem X, si el Escribir_item(X) final en el plan S_1 es ejecutado por T_i, entonces el Escribir_item(X) final en el plan S_2 debe ser ejecutada por T_i.

Secuencialidad en cuanto a vistas

Cada operación de lectura en S_2 debe leer el mismo valor que en S_1. Además, el último valor escrito en la BD debe ser el mismo en ambos planes.

Plan S_1

T_1	T_2
Leer_item(X)	
X := X–N	
Escribir_item(X)	
	Leer_item(X)
	X := X+M
	Escribir_item(X)
Leer_item(Y)	
Y := Y+N	
Escribir_item(Y)	

Plan S_2

T_1	T_2
Leer_item(X)	
X := X–N	
Escribir_item(X)	
Leer_item(Y)	
Y := Y+N	
Escribir_item(Y)	
	Leer_item(X)
	X := X+M
	Escribir_item(X)

El plan S_1 es equivalente al plan S_2, por tanto, S_1 es secuenciable en cuanto a vistas.

Secuencialidad en cuanto a vistas

Cada operación de lectura en S_2 debe leer el mismo valor que en S_1. Además, el último valor escrito en la BD debe ser el mismo en ambos planes.

Plan S_1

T_1	T_2
Leer_item(X)	
X := X–N	
	Leer_item(X)
	X := X+M
Escribir_item(X)	
Leer_item(Y)	
Y := Y+N	
	Escribir_item(X)
Escribir_item(Y)	

Plan S_2

T_1	T_2
Leer_item(X)	
X := X–N	
Escribir_item(X)	
Leer_item(Y)	
Y := Y+N	
Escribir_item(Y)	
	Leer_item(X)
	X := X+M
	Escribir_item(X)

Viola regla 1 porque no lee el valor escrito por T_1.

Planificaciones recuperables

- Puede que en una planificación S una transacción T_1 lea datos producidos por otra transacción T_2.
- Si la transacción T_2 termina y confirma y luego la transacción T_1 aborta, entonces la planificación S se vuelve irrecuperable ya que no pueden deshacerse los cambios de la transacción T_2.
- Para que una planificación sea recuperable la confirmación de una Transacción T_j que ha leído datos de una transacción T_i debe ser postergada por el SGBD hasta que la transacción T_i confirme.

T1	T2
Leer_item(X)	
X := X–N	
Escribir_item(X)	
	Leer_item(X)
	X := X+M
	Escribir_item(X)
	Commit
Fallo	

Plan no recuperable

T1	T2
Leer_item(X)	
X := X–N	
Escribir_item(X)	
	Leer_item(X)
	X := X+M
	Escribir_item(X)
	Confirmar luego de T_1
Fallo	

Plan recuperable

Retrocesos en Cascada

- El fallo de una transacción cuyos efectos han sido observados por otras transacciones produce que todas las transacciones sean retrocedidas.
- Considere la siguiente planificación, en la cual ninguna transacción ha confirmado:

T_{10}	T_{11}	T_{12}
leer (A)		
leer (B)		
escribir (A)		
	leer (A)	
	escribir (A)	
		leer (A)

- Si T_{10} falla debe ser retrocedida, provocando que T_{11} y T_{12} también sean retrocedidas.
- Los retrocesos en cascada pueden ocasionar una sobrecarga al sistema cuando los cambios a deshacer son demasiados.
- Una forma de evitar los retrocesos en cascada seria retrasar la lectura de los datos cambiados por una transacción hasta que esta confirme.
  - Sin embargo este enfoque reduce el grado de concurrencia.

Prueba de secuencialidad

Grafos de precedencia

Para verificar la secuencialidad, se construye un grafo de precedencia para el plan de ejecución.

G(N,E) es un grafo dirigido, donde:

N = $\{T_1, T_2, ..., T_n\}$ ... conjunto de nodos;
E = $\{e_1, e_2, ..., e_m\}$ ... conjunto de arcos dirigidos.

- Existe un nodo por cada transacción T_i en el plan.
- Cada arco e_i es de la forma T_j -> T_k, 1<=j<=n, 1<=k<=n, donde T_j es el nodo inicial de e_i y T_k es el nodo final.

Prueba de secuencialidad

- Para cada transacción T_i del plan S
  - Crear un nodo etiquetado T_i en el grafo.
  - Para todo par (Ti, Tj), crear un arco para cada caso en el que se tiene que:

Operaciones	**Arco**
T_i = escribir_item(X) precede a T_j = leer_item(X)	T_i -> T_j
T_i = leer_item(X) precede a T_j = escribir_item(X)	T_i -> T_j
T_i = escribir_item(X) precede a T_j = escribir_item(X)	T_i -> T_j

- El plan S es secuenciable si el grafo de precedencias no tiene ciclos.

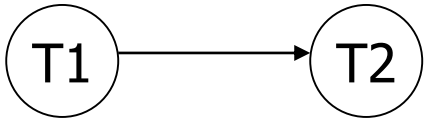
Prueba de secuencialidad

T1	T2
Leer_item(X)	
X := X–N	
Escribir_item(X)	
	Leer_item(X)
	X := X+M
	Escribir_item(X)
Leer_item(Y)	
Y := Y+N	
Escribir_item(Y)	

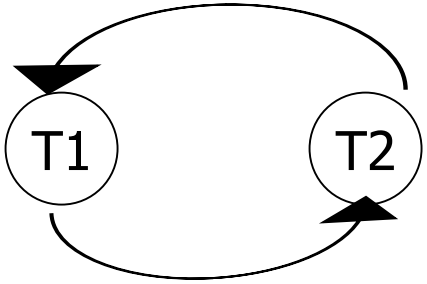
Plan D

T1	T2
Leer_item(X)	
X := X–N	
	Leer_item(X)
	X := X+M
Escribir_item(X)	
Leer_item(Y)	
Y := Y+N	
	Escribir_item(X)
Escribir_item(Y)	

Plan C

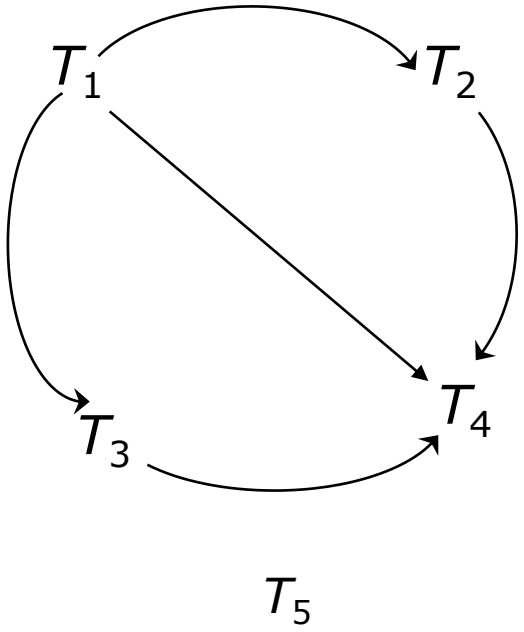


Contiene T1 -> T2 porque
T1 ejecuta Leer_item(X) antes que
T2 ejecute Escribir_item(X)
Contiene T2 -> T1 porque
T2 ejecuta Leer_item(X) antes que
T1 ejecute Escribir_item(X)



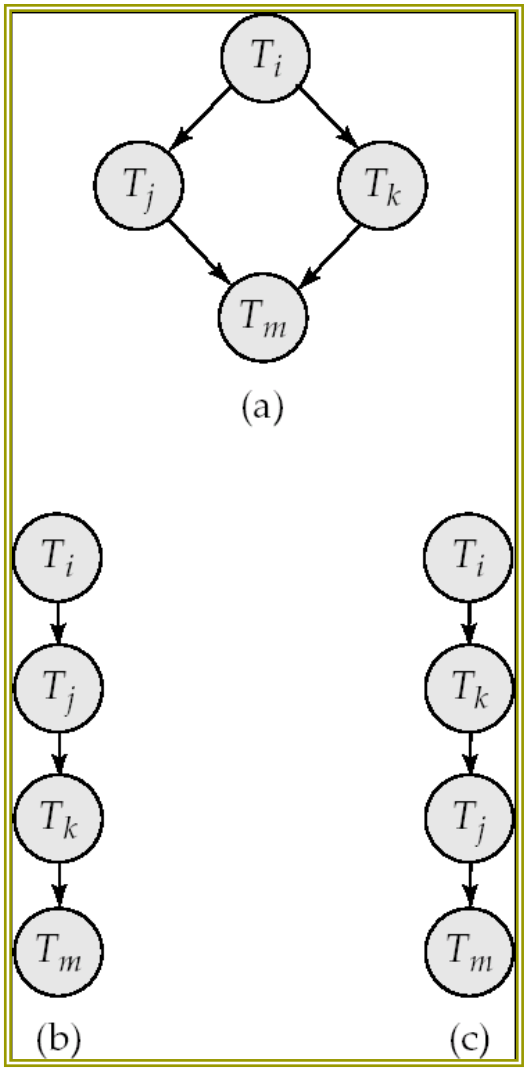
Prueba de secuencialidad

T_1	T_2	T_3	T_4	T_5
	read(X)			
read(Y)				
read(Z)				
				read(V)
				read(W)
				read(W)
	read(Y)			
	write(Y)			
		write(Z)		
read(U)				
			read(Y)	
			write(Y)	
			read(Z)	
			write(Z)	
read(U)				
write(U)				



Prueba de secuencialidad

- Una planificación es secuenciable en cuanto a conflictos si su grafo de precedencias es acíclico.
- Pueden utilizarse algoritmos de detección de ciclos para determinar si hay conflicto o no.
  - Lo cual tiene un costo $O(n^2)$, donde n es el número de transacciones.
- Si el grafo es acíclico un orden secuencial puede determinarse mediante un ordenamiento topológico.
  - Para el ejemplo uno de los ordenamientos es $T_5 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_4$



Control de Concurrencia

- Un SGBD debe proveer un mecanismo que asegure planificaciones:
  - Secuenciales en cuanto a vistas o a conflictos
  - Recuperables y preferiblemente sin cascadas
- Una política de que solo se ejecute una transacción a la vez cumple las restricciones anteriores, pero no permite concurrencia
- Probar que las transacciones son secuenciables luego de que ocurren puede ser otro enfoque.
  - Pero podría retardase mucho la confirmación de algunas transacciones.
- Una mejor estrategia sería utilizar protocolos de control de concurrencia que aseguren la secuencialidad, recuperabilidad y evite los retrocesos en cascada.
- Los protocolos de control de concurrencia pueden proveer diferentes niveles de concurrencia, sin embargo aumentan la carga del sistema para realizar su tarea.
- Las pruebas de secuencialidad ayudan probar que un protocolo de control de concurrencia sea correcto.

Protocolos para Control de Concurrencia

Los esquemas de control de concurrencia aseguran que se conserve la propiedad de aislamiento entre transacciones concurrentes. A continuación se describen esquemas basados en la secuencialidad:

- La mayoría de los protocolos utilizan **técnicas de bloqueo** (locking) de los ítems de datos para prevenir que múltiples transacciones accedan a los mismo ítems concurrentemente.
- Otros métodos se basan en el empleo de **marcas de tiempo** (timestamping) que identifica unívocamente a cada transacción. Estas marcas son generadas por el sistema, de manera que las transacciones pueden ser ordenadas según sus marcas para asegurar la secuencialidad.
- Los **protocolos optimistas** suponen que no habrá problemas de concurrencia, y si los hay, se corrigen sus efectos.

Protocolos basados en el Bloqueo

Una forma de asegurar la secuencialidad es mediante la exclusión mutua, es decir, mientras una transacción accede a un elemento de datos, ninguna otra puede modificar dicho elemento.

Modos de bloqueos

Compartido (lectura)

- Más de una transacción puede adquirir un bloqueo de lectura sobre un recurso.
- Una transacción que tiene bloqueo de lectura sobre un recurso no puede modificarlo, solo leerlo.
- Ninguna otra transacción puede leer el recurso sin bloquearlo en modo de lectura (en Oracle si).
- Ninguna transacción puede modificar los datos.
- Sirve para garantizar una estabilidad en el valor leído mientras alguna transacción mantenga un bloqueo.

Exclusivo (escritura)

- Solo la transacción que adquiere el bloqueo puede modificar el dato.
- Ninguna otra transacción puede ni siquiera leer el dato (en Oracle si).

Concesión de bloqueos

- Toda transacción debe solicitar los bloqueos al ***gestor de control de concurrencia***.
- El gestor puede conceder un bloqueo sobre un mismo dato a dos transacciones distintas cuando los modos son compatibles.
  - Si T_2 posee un bloqueo en modo compartido sobre un elemento de datos, y T_1 solicita bloqueo exclusivo sobre el mismo dato, entonces T_1 debe esperar a que T_2 libere el bloqueo.
  - Si T_3 solicita bloqueo en modo compartido antes que T_2 termine, el gestor puede conceder el bloqueo porque es compatible, y así sucesivamente.
  - Si los bloqueos compatibles son siempre concedidos postergando la concesión de bloqueos incompatibles, T_1 podría quedar en estado de **inanición.**

Evitando la Inanición

- Se concede un bloqueo a T_i si:
  - El dato no esta bloqueado en un modo conflictivo, y
  - No existe otra transacción que esté esperando un bloqueo sobre el dato y lo haya solicitado antes que T_i.

Interbloqueos

- El **interbloqueo** o **bloqueo mortal** (**deadlock**) ocurre cuando dos o más transacciones están esperando simultáneamente por datos que están siendo bloqueados por una de las otras transacciones.
- Ejemplo, T_0 espera a un elemento de dato que posee T_1, T_1 espera a un elemento de dato que posee T_2, y T_2 espera por el elemento de dato que posee T_0.

T1	T2
update EMP **set sal=sal*1.1** **where id=1**	
	update EMP **set sal=sal*1.1** **where id=2**
update EMP **set sal=sal*1.1** **where id=2**	Bloqueado por T2
	update EMP **set sal=sal*1.1** **where id=1** — Bloqueado por T1

Ejemplo

- **T1**: quiere cambiar salario de empleado 1 y luego de empleado 2.
- **T2**: quiere cambiar salario de empleado 2 y luego de empleado 1

Interbloqueos

- Hay dos métodos para tratar el problema:
  - Protocolo de prevención de interbloqueos
  - Esquema de detección y recuperación de interbloqueos.

Protocolo de prevención de bloqueos

- Requiere que cada transacción bloquee todos los ítems que va a necesitar.
- Si alguno de ellos no puede ser obtenido, entonces ninguno de los demás ítems es bloqueado, y la transacción pasa a un estado de espera.
- La transacción vuelve a intentar posteriormente obtener el bloqueo de todos los ítems que necesita.

Interbloqueos

Esquema de detección de interbloqueos

- Se ejecutan las transacciones y periódicamente el sistema chequea si existe algún bloqueo mortal.
- Una forma de detectar un estado de bloqueo mortal es construyendo un **grafo de esperas**
  - Se crea un nodo por cada transacción que se ejecuta en el plan.
  - Cuando una transacción Ti está esperando a que se libere el ítem X bloqueado por otra transacción Tj, se dibuja un arco (Ti -> Tj).
  - Se dibuja un arco por cada transacción Tj que tenga bloqueado el dato.
  - Existe un bloqueo mortal si y solo si el grafo de esperas tiene un ciclo.
- Si se detecta un bloqueo mortal, alguna de las transacciones causantes debe ser abortada, y cualquier efecto sobre la BD debe ser deshecho (rollback).
- Para detectar los interbloqueos se utiliza un algoritmo de detección de ciclos.
  - El sistema debe correr este algoritmo cada cierto tiempo.
  - Cada cuanto?

Recuperación de Interbloqueos

- Cuando se detecta un interbloqueo:
  - Alguna transacción debe ser retrocedida (rollback) para romper el deadlock.
    - Se debe seleccionar la transacción tal que se incurra en el mínimo costo.
  - El retroceso pude ser:
    - Total: La transacción es abortada, retrocedida y reiniciada.
    - Parcial: La transacción es retrocedida parcialmente hasta un punto tal que se rompa el interbloqueo.
  - Puede ocurrir estados de Inanición cuando un transacción es repetidamente elegida para recuperarse de un interbloqueo.
    - Para evitar esto se debe llevar una cuenta del número de retrocesos como factor de costo de manera a no elegir la misma transacción repetidamente.

Protocolo de Bloqueo en Dos Fases

- Este protocolo exige que cada transacción realice sus peticiones de bloqueo y desbloqueo en dos fases:
  - **Fase de crecimiento:** La transacción puede obtener bloqueos pero no liberarlos.
  - **Fase de descrecimiento:** La transacción puede liberar bloqueos pero no obtener ninguno nuevo.
- Inicialmente una transacción esta en fase de crecimiento, una vez que libere un bloqueo entra en fase de decrecimiento.
- **Este protocolo asegura la secuencialidad en cuanto a conflictos.**
  - Se llama **punto de bloqueo** al momento en que una transacción obtiene su bloqueo final.
  - La ordenación de los puntos de bloqueo determina un orden de secuencialidad.
- Este protocolo no evita los interbloqueos ni retrocesos en cascada.

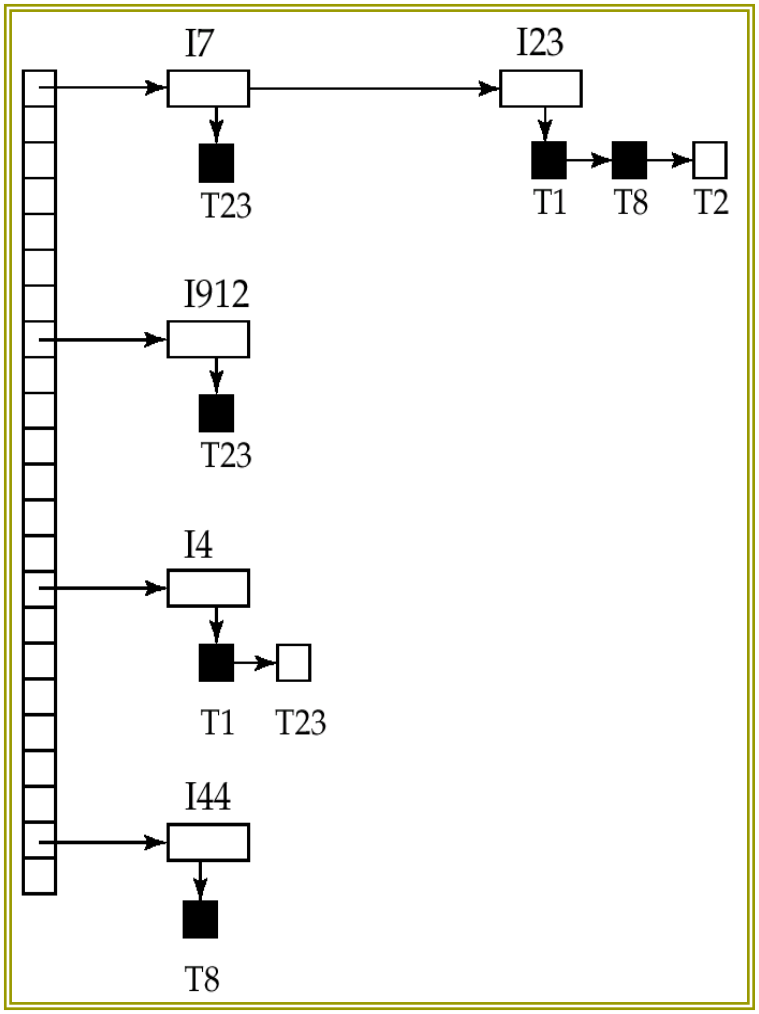
Protocolo de Bloqueo en Dos Fases

- **Protocolo de bloqueo estricto en dos fases**
  - Los bloqueos exclusivos se liberan al final de la transacción
  - Se evitan los retrocesos es cascada.
- **Protocolo de bloqueo riguroso de dos fases**
  - Todos los bloqueos se liberan al final de la transacción.
  - Las transacciones se pueden secuenciar en el orden que comprometen.
- **Conversiones de Bloqueo**
  - Permite convertir un bloqueo compartido a exclusivo y viceversa.
  - En la fase de crecimiento se permiten conversiones de compartido a exclusivo (**subir**).
  - En la fase de decrecimiento se permite conversiones de exclusivo a compartido (**bajar**).
- **Gestión automática de bloqueos.**
  - Cuando una transacción realiza una operación de **Leer** se genera automáticamente una petición de **bloqueo compartido.**
  - Cuando una transacción realiza una operación de **Escribir** se genera automáticamente una petición de **bloqueo exclusivo** o una operación **Subir**
  - Los bloqueos se liberan una vez que la transacción confirma o aborte.

Gestor de Bloqueos

- Un Gestor de bloqueos puede ser implementado como un proceso separado que recibe peticiones de bloqueo y desbloqueo
- El gestor de bloqueo responde un mensaje de **petición** con un mensaje de **concesión.**
- Una transacción que ha solicitado un bloqueo debe esperar hasta recibir el mensaje de concesión o un mensaje de retroceso.
- El gestor de bloqueo mantiene una **tabla de bloqueos** que almacena los bloqueos concedidos y las peticiones solicitadas.
- La tabla de bloqueos es una tabla hash en memoria cuyas entradas corresponden a los elementos de datos bloqueados.
- Por cada elementos de datos hay una lista enlazada con registros que representan solicitudes concedidas (pero no liberadas) y solicitudes pendientes.

Tabla de Bloqueos



- Los rectángulos negros indican bloqueos concedidos y los blancos solicitudes pendientes
- Los registros de bloqueos también indican el tipo de bloqueo
- Las nuevas peticiones son agregadas al final de la lista y concedidas si son compatibles con los anteriores.
- Las peticiones de desbloqueo eliminan una entrada de la lista y provocan que se conceda el o los siguientes bloqueos si son compatibles con los existentes.
- Si una transacción aborta, todos los bloqueos concedidos a ésta son eliminados de la tabla.
  - El Gestor de bloqueos también lleva una lista de los elementos de datos bloqueados por cada transacción.

Protocolos basados en Marcas de Tiempo

- Una marca de tiempo es un identificador único creado por el gestor de la BD para identificar una transacción.
- Los valores de las marcas de tiempo se asignan por el orden en el cual las transacciones son recibidas por el sistema.
- La marca de tiempo puede ser considerada como el inicio de una transacción.
- Las marcas de tiempo pueden ser generadas de varias formas:
  - Utilizando un contador,
  - La hora del reloj del sistema.
- El método consiste en ordenar las transacciones según sus marcas de tiempo.
  - Si un transacción T_i ingresa al sistema recibe una marca de tiempo $MT(T_i)$.
  - Si luego entra una transacción T_j el sistema genera una marca de tiempo $MT(T_j)$, tal que $MT(T_i) < MT(T_j)$
  - El sistema debe asegurar que se produzcan planificaciones equivalentes a una planificación secuencial en la cual T_i aparece antes que T_j

Protocolos basados en Marcas de Tiempo

Ordenación por Marcas de Tiempo

- Se permite que las transacciones se ejecuten libremente.
- Hay que asegurar de que por cada ítem accedido por más de una transacción en el plan, el orden de acceso no viole la secuencialidad, para lo cual se asocian a cada ítem X de la BD dos marcas de tiempo:
  - MT_lectura(X): Es la mayor entre todas las marcas de tiempo de las transacciones que sucesivamente han leído X.
  - MT_escritura(X): Es la mayor entre todas las marcas de tiempo de las transacciones que sucesivamente han escrito X.
- Estas marcas temporales se actualizan cada vez que se ejecuta una nueva operación leer o escribir.

Protocolo de Ordenación por Marcas de Tiempo

El algoritmo de control de concurrencia debe comprobar si la ordenación por marcas de tiempo es violada cuando la transacción **T** intenta realizar una operación de lectura o escritura.

La transacción T intenta una operación Escribir_item(X)	
a) Si MT_lectura(X) > MT(T)	Abortar y deshacer la transacción T. Alguna transacción posterior a T ya leyó el ítem X antes de que T realice la escritura. Se viola la secuencialidad.
b) Si MT_escritura(X) > MT(T)	Abortar y deshacer la transacción T. Alguna transacción posterior a T ya escribió el valor de X. Se evita el Escribir_item(X) de T para que no modifique el valor correcto con uno obsoleto.
SI no se cumplen a) y b)	Ejecutar la operación Escribir_item(X) de T y hacer MT_escritura <- MT(T). La Transacción T llega a tiempo para realizar la operación
La transacción T intenta una operación Leer_item(X)	
a) Si MT_escritura(X) > MT(T)	Abortar y deshacer la transacción T. Alguna Transacción posterior a T ya sobrescribió el valor de X que T debió leer.
SI no se cumple a)	Ejecutar la operación Leer_item(X) de T y asignar a MT_lectura (X) el valor más grande entre MT(T) y MT_lectura(X).

Protocolo de Ordenación por Marcas de Tiempo

- El protocolo de ordenación por marcas temporales garantiza la secuencialidad en cuanto a conflictos.
  - Se asegura que todas las operaciones con marca de tiempo menor se ejecuten antes de las de marcas de tiempo mayor.
- Se asegura la ausencia de interbloqueos ya que ninguna transacción queda bloqueada antes de realizar alguna operación.
- Existe la posibilidad de inanición ya que transacciones largas puede sufrir de repetidos reinicios.
  - Si se detecta el reinicio repetido alguna transacción, el sistema debe bloquear las transacciones conflictivas para permitir que la transacción reiniciada termine.
- El protocolo de marcas de tiempo puede generar planificaciones no recuperables.
  - La recuperabilidad y ausencia de cascadas se puede asegurar realizando escrituras atómicas al final de la transacción.
  - La recuperabilidad y ausencia de cascadas se puede asegurar bloqueando las lecturas de los elementos no comprometidos..
  - La recuperabilidad se puede asegurar posponiendo el compromiso de una transacción hasta que comprometan todas las transacciones que hayan previamente modificado los datos leídos por la primera.

Regla de escritura de Thomas

La regla de escritura de Thomas modifica el protocolo de marcas temporales para asegurar la secuencialidad en cuanto a vistas permitiendo una mayor concurrencia.

La transacción T intenta una operación Escribir_item(X)	
a) Si MT_lectura(X) > MT(T)	Abortar y deshacer la transacción T. Alguna transacción posterior a T ya leyó el ítem X antes de que T realice la escritura. Se viola la secuencialidad.
b) Si MT_escritura(X) > MT(T)	**Ignorar la operación Escribir_item(X) de T**. Alguna transacción posterior a T ya escribió el valor de X. Se ignora el Escribir_item(X) de T para que no modifique el valor correcto con uno obsoleto.
SI no se cumplen a) y b)	Ejecutar la operación Escribir_item(X) de T y hacer MT_escritura <- MT(T). La Transacción T llega a tiempo para realizar la operación
La transacción T intenta una operación Leer_item(X)	
a) Si MT_escritura(X) > MT(T)	Abortar y deshacer la transacción T. Alguna Transacción posterior a T ya sobrescribió el valor de X que T debió leer.
SI no se cumple a)	Ejecutar la operación Leer_item(X) de T y asignar a MT_lectura (X) el valor más grande entre MT(T) y MT_lectura(X).

Protocolos Basados en Validación

- Se considera que las transacciones se ejecutan en tres fases
  - **Fase de Lectura:** La transacción se ejecuta leyendo los datos que necesite almacenando los valores en variables locales. Todas las operaciones de escritura se realizan en la variables locales.
  - **Fase de Validación:** La transacción realiza una prueba de validación para determinar si se pueden escribir las variables locales a disco sin afectar la secuencialidad.
  - **Fase de Escritura:** Si la validación tiene éxito se escriben los datos a disco. En otro caso se retrocede la transacción.
- Todas las transacciones deben ejecutar las tres fases en orden. Sin embargo en transacciones concurrentes, las tres fases pueden entrelazarse.
- Para realizar la validación se necesita conocer el momento en que tienen lugar cada fase de cada transacción. A cada transacción T se asocian tres marcas temporales
  - **Inicio(T)**: Momento en que T inicia su ejecución.
  - **Validación(T)**: Momento en que T inicia la fase de validación.
  - **Fin(T)**: Momento en que T termina su fase de escritura.
- Se utiliza como marca temporal de T al momento de su validación.
  - MT(T) = Validación(T)

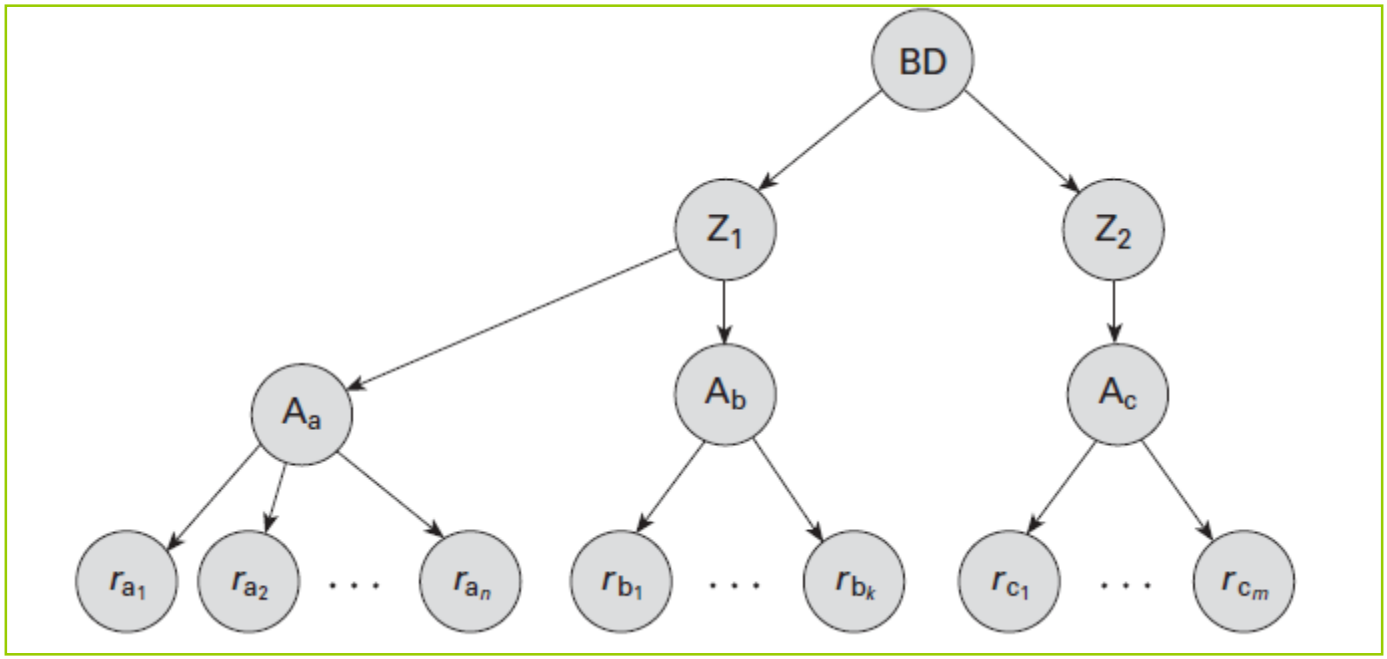
Protocolos Basados en Validación

- La **Validación** de un transacción Ti consiste en comprobar que para toda transacción Tj, tal que MT(Ti) < MT(Tj) se cumplan una de las condiciones:
  - **Fin(Ti) < Inicio(Tj)**. El orden de ejecución entre las transacciones es secuencial.
  - **Inicio(Tj) < Fin(Ti) < Validación(Tj) y que las escrituras de Ti no tienen elementos de datos en común con las lecturas de Tj**. Las escrituras de Ti no afectan a Tj y Tj no interfiere con Ti, por lo tanto se mantiene el orden secuencial.
- Este esquema evita los retrocesos en cascada debido a que las escrituras reales solo tienen lugar si se pasa la validación.
- Es posible que las transacciones largas sufran de inanición.
  - La solución a este problema es la misma que se aplica en el protocolo de marcas temporales.
- Este esquema de control de concurrencia es apropiado para ser aplicados en sistemas donde las transacciones son mayormente de solo lectura.
  - La sobrecarga impuesta por el mismo es pequeña.
  - Las transacciones no necesitan ser bloqueadas y se ejecutan libremente.
- Este esquema se denomina **Control de Concurrencia Optimista** ya que las transacciones se ejecutan asumiendo que validarán al final.

Granularidad de los Ítems de Datos

- La granularidad determina el nivel de bloqueo de datos.
- Dependiendo del DBMS será posible bloquear a nivel de:
  - Campo
  - Registro
  - Tabla
  - BD
- Una granularidad más fina supone un mayor grado de concurrencia
  - Pero también mucha sobrecarga en tiempo de ejecución y de espacio ocupado con información sobre los bloqueos en el Gestor de Concurrencia.
- Normalmente los DMBS no soportan el bloqueo de campos.
- Los datos en los niveles de granularidad disponibles pueden ser representados por un **árbol de granularidad**.
  - Cuando se bloquea una parte del árbol, automáticamente se bloquean los niveles descendientes.
  - De la misma forma no se puede bloquear un nodo del árbol si alguno de sus nodos ascendientes ya se encuentra bloqueado por otra entidad.
    - Se recorre el árbol desde la raíz hacia el nodo a bloquear si se encuentra un nodo ascendiente bloqueado en modo incompatible entonces se retrasa la petición de bloqueo.

Granularidad de los Ítems de Datos



- Los niveles en este árbol son:
  - BD (Grueso)
  - Zona
  - Archivo
  - Registro (Fino)

Bloqueo Intencional

- Cuando se bloquea explícitamente un nodo del árbol de granularidad, implícitamente se bloquean sus descendientes.
  - Cuando alguna transacción desea bloquear un nodo descendiente, se recorre desde la raíz y si encuentra un nodo ascendiente bloqueado se debe esperar.
- Pero, como saber si un nodo ascendiente puede ser bloqueado ?
  - Solo se puede bloquear un nodo si sus descendientes no están bloqueados.
  - Ej.: No se puede bloquear una tabla si alguna de sus filas esta bloqueada por otra transacción.
- Cuando existe granularidad de bloqueos, se utilizan tres modos de bloqueos adicionales:

- **Intencional Compartido (IC)**: Indica que existe un bloqueo explicito compartido en algún nodo descendiente.
- **Intencional Exclusivo (IX)**: Indica que existe un bloqueo explicito compartido o exclusivo en algún nodo descendiente.
- **Intencional Compartido y Exclusivo (ICX)**: Indica que el subárbol del nodo que se está bloqueando intencionalmente se bloquea explícitamente en modo compartido y algún nodo descendiente se está bloquea explícitamente en modo exclusivo.

	IC	IX	C	IXC	X
IC	cierto	cierto	cierto	cierto	falso
IX	cierto	cierto	falso	falso	falso
C	cierto	falso	cierto	falso	falso
IXC	cierto	falso	falso	falso	falso
X	falso	falso	falso	falso	falso

Bloqueo Intencional

- Una Transacción Ti puede bloquear un nodo Q, usando las siguientes reglas:
  - Debe observarse la matriz de compatibilidad.
  - El nodo raíz debe ser bloqueado en primer lugar y puede ser bloqueado en cualquier modo.
  - Un nodo Q puede ser bloqueado en modo C o IC solo si se está bloqueando el nodo padre de Q en modo IX o IC.
  - Un nodo Q puede ser bloqueado en modo X, IXC o IX solo si se esta bloqueando el nodo padre de Q en modo IX o IXC.
  - Se puede bloquear un nodo solo si no se ha desbloqueado un nodo anteriormente (protocolo de dos fases).
  - Puede desbloquear un nodo Q solo si no se ha bloqueado ninguno de los descendientes de Q.
- Lo bloqueos deben ser adquiridos en orden descendiente al explorar el árbol y se liberan en orden ascendiente.

Bloqueo Intencional

- La transacción T_1 lee el registro r_{a2} del archivo A_a. Entonces T_1 necesita bloquear la base de datos, la zona Z_1 y el A_a en modo IC (y en este orden) y finalmente debe bloquear r_{a2} en modo C.
- La transacción T_2 modifica el registro r_{a9} del archivo A_a. Entonces T_2 necesita bloquear la base de datos, la zona Z_1 y el archivo A_a en modo IX y finalmente debe bloquear r_{a9} en modo X.
- La transacción T_3 lee todos los registros del archivo A_a. Entonces, T_3 necesita bloquear la base de datos y la zona Z_1 (y ello en este orden) en modo IC y finalmente debe bloquear A_a en modo C.
- La transacción T_4 lee toda la base de datos. Puede hacer esto una vez que bloquee la base de datos en modo C.

- Las transacciones T_1, T_3 y T_4 pueden acceder concurrentemente a la BD.
- La Transacción T_2 se puede ejecutar concurrentemente con T_1 pero no con T_3 ni con T_4.