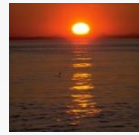
Capítulo 15: Transacciones

Fundamentos de Bases de datos, 5ª Edición.

Capítulo 15: Transacciones

- Concepto de transacción
- Estados de una transacción
- Ejecuciones concurrentes
- Secuencialidad
- Recuperabilidad
- Implementación del aislamiento
- Definición de transacción en SQL
- Comprobación de la secuencialidad



Concepto de una transacción

- Una ***transacción*** es una *unidad* de ejecución de programa que accede, y posiblemente actualiza, a varios elementos de datos.
- Una transacción debe ver una base de datos consistente.
- Durante la ejecución de la transacción la base de datos puede ser inconsistente.
- Cuando se compromete una transacción la base de datos deber ser consistente.
- Se pueden ejecutar múltiples transacciones en paralelo.
- Dos enfoques principales a tener en cuenta:
  - Fallos de varias clases, tales como fallos de hardware y caídas del sistema
  - Ejecución concurrente de múltiples transacciones

Propiedades ACID

Una **transacción** es una unidad de ejecución de un programa que accede y posiblemente actualiza varios elementos de datos. Para preservar la integridad de los datos, el sistema de bases de datos debe asegurar:

- **Atomicidad.** O todas las operaciones de la transacción se reflejan correctamente en la base de datos, o ninguna.
- **Consistencia.** La ejecución de una transacción en aislamiento preserva la consistencia de la base de datos..
- **Aislamiento.** Aunque varias transacciones se pueden ejecutar concurrentemente, cada transacción debe ignorar a las otras transacciones que se ejecutan concurrentemente con ella. Los resultados de las transacciones intermedias deben ocultarse de otras transacciones ejecutadas concurrentemente.
  - Es decir, por cada par de transacciones T_i y T_j, le parece a T_i, que, o bien T_j ha terminado su ejecución antes de que comience T_i , o que T_j ha comenzado su ejecución después de que T_i terminara.
- **Durabilidad.** Tras la finalización con éxito de una transacción permanecen los cambios realizados en la base de datos, incluso si hay fallos en el sistema.

Ejemplo de transferencia de fondos

- Transacción para transferir 50 € desde una cuenta *A* a una cuenta *B*:
  1. **leer**(*A*)
  2. *A* := *A* – 50
  3. **escribir**(*A*)
  4. **leer**(*B*)
  5. *B* := *B* + 50
  6. **escribir**(*B*)
- **Requisito de atomicidad** – si la transacción falla después del paso 3 y antes del paso 6, el sistema debería asegurar que sus actualizaciones no se reflejan en la base de datos, de lo contrario resultará una inconsistencia.
- **Requisito de consistencia** – la suma de *A* y *B* no se altera por la ejecución de la transacción.

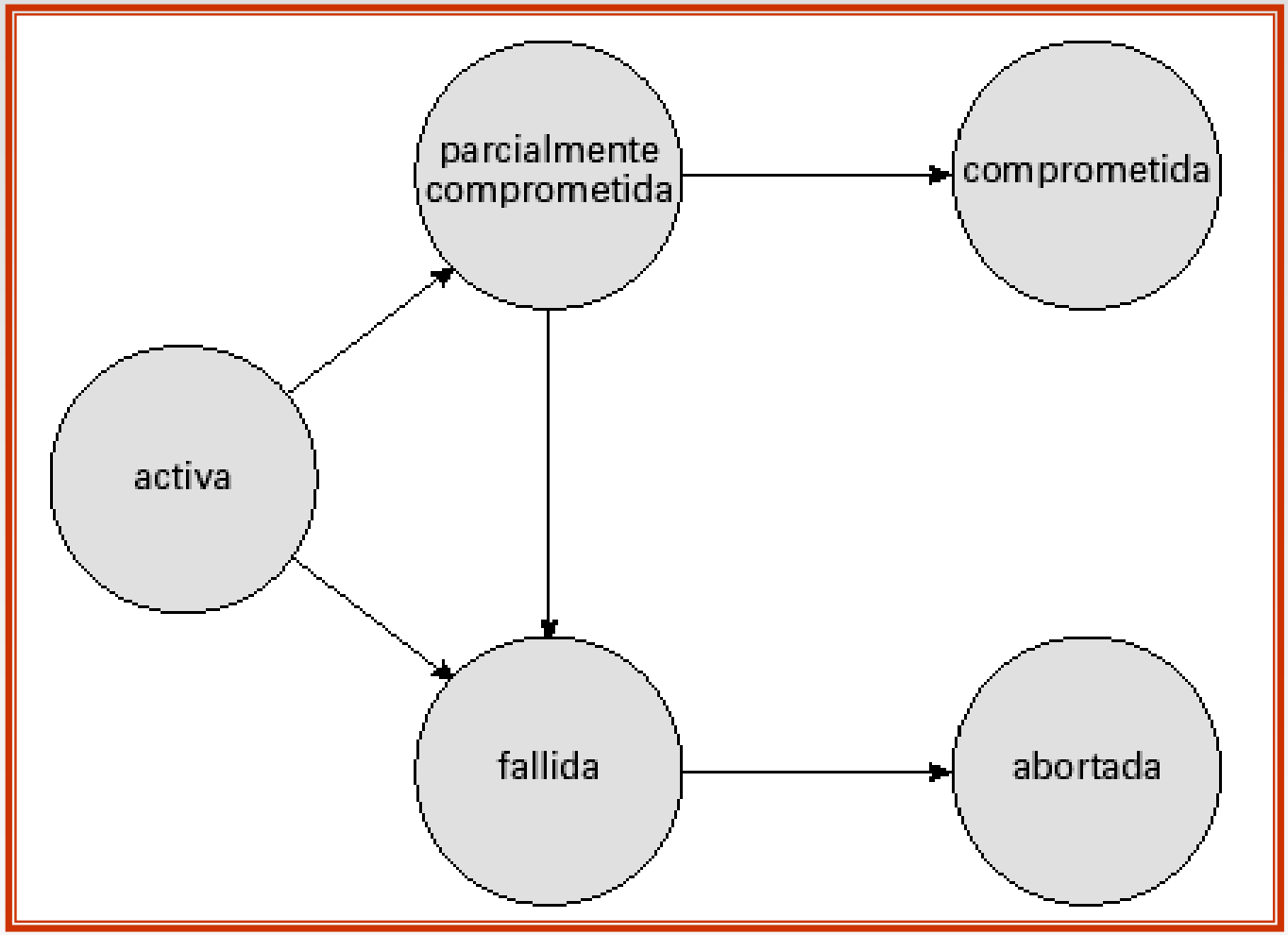
Ejemplo de transferencia de fondos (Cont.)

- **Requisito de aislamiento** – si entre los pasos 3 y 6 se permite acceder a otra transacción a la base de datos parcialmente actualizada, verá una base de datos inconsistente (la suma de *A* + *B* será menor de lo que debería).
  - El aislamiento se puede asegurar de forma trivial ejecutando las transacciones ***secuencialmente***, es decir, una detrás de otra.
  - Sin embargo la ejecución de diversas transacciones concurrentemente tiene, como se verá, significativos beneficios.
- **Requisito de durabilidad** – desde que se notifica al usuario que se ha completado la transacción (es decir, que ha tenido lugar la transferencia de 50 €), las actualizaciones de la base de datos producidas por la transacción deben permanecer, a pesar de los fallos.

Estados de la transacción

- **Activa**, el estado inicial; la transacción permanece en este estado mientras se está ejecutando
- **Parcialmente comprometida**, después que se ha ejecutado la instrucción final.
- **Fallida**, después de descubrir que la ejecución normal ya no puede llevarse a cabo.
- **Abortada**, después que la transacción se ha retrocedido y la base de datos restaurado a su estado anterior al inicio de la transacción. Dos opciones después de que haya abortado:
  - reiniciar la transacción – sólo si no hay errores lógicos internos
  - cancelar la transacción
- **Comprometida**, después de terminación con éxito.

Estado de la transacción (cont.)

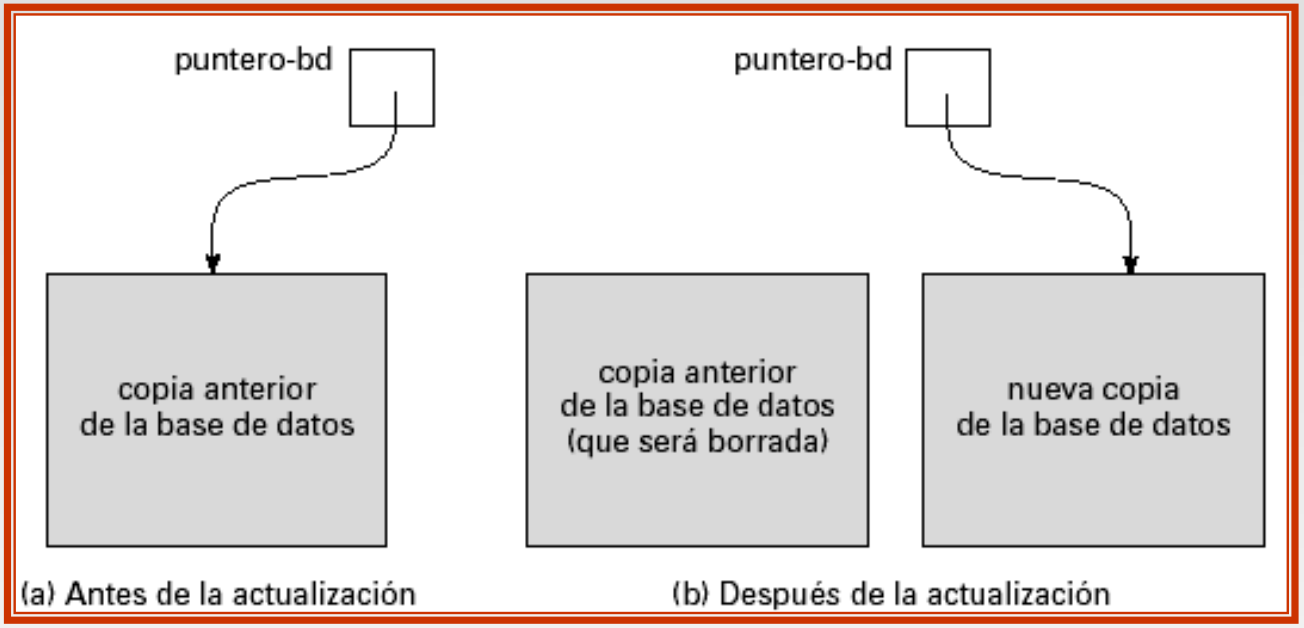


Implementación de atomicidad y durabilidad

- El componente para la **gestión de la recuperación** de un sistema de bases de datos implementa el soporte para la atomicidad y durabilidad.
- El esquema de la *base de datos en la sombra*:
  - asume que sólo está activa una transacción en cada momento.
  - un puntero, denominado puntero_db, siempre apunta a la copia consistente actual de la base de datos.
  - todas las actualizaciones son hechas sobre una copia en la sombra de la base de datos, y el **puntero_db** apunta a la copia en la sombra actualizada, sólo después que la transacción alcance un compromiso parcial y todas las páginas actualizadas se hayan desviado a disco.
  - en caso de fallo en la transacción se puede usar la antigua copia consistente apuntada por **puntero_db** y se puede borrar la copia en la sombra.

Implementación de atomicidad y durabilidad (cont.)

Esquema de la base de datos en la sombra:



- Se asume que los discos no fallan
- Útil para los editores de textos, pero
  - Extremadamente ineficiente para grandes bases de datos (¿por qué?)
  - No permite a las transacciones ejecutarse concurrentemente
- Se verán mejor los esquemas en el Capítulo 17.

Ejecuciones concurrentes

- Se permite ejecutar concurrentemente varias transacciones en el sistema. Las ventajas son:
  - **aumento de la utilización del procesador y del disco**, conduciendo a mejorar la *productividad* de la transacción una transacción puede estar utilizando la CPU, mientras otras está leyendo desde o escribiendo a disco
  - **reducción del tiempo medio de respuesta** de la transacciones: Las transacciones pequeñas no tienen necesidad de esperar detrás de las grandes.
- **Esquemas de control de concurrencia** - mecanismos para conseguir aislamiento, es decir, para controlar la interacción entre la transacciones concurrentes a la hora de impedir que destruyan la consistencia de la base de datos
  - Se estudia en el Capítulo 16, después de estudiar la noción de exactitud de las ejecuciones concurrentes.

Planificaciones

- **Planificaciones** – secuencias que indican el orden cronológico en el que se ejecutan las instrucciones de las transacciones concurrentes
  - una planificación para un conjunto de transacciones debe constar de todas las instrucciones de estas transacciones
  - debe preservar el orden en el que aparecen las instrucciones en cada transacción individual.
- Una transacción que se completa correctamente tendrá como última sentencia instrucciones de compromiso (se omite si es obvio)
- Una transacción que falla en su ejecución tendrá una instrucción de recuperación como última sentencia (se omite si es obvio)

Planificación 1

- Sea T_1 la transferencia de 50€ desde *A* a *B*, y T_2 la transferencia del 10% del saldo desde *A* a *B.*
- Una planificación secuencial en la que T_1 es seguida de T_2.

T_1	T_2
leer(A)	
$A := A$ - 50	
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	
	leer(A)
	$temp := A * 0,1$
	$A := A$ - $temp$
	escribir(A)
	leer(B)
	$B := B + temp$
	escribir(B)

Planificación 2

- Una planificación secuencial donde *T2* es seguido de *T*1

T_1	T_2
	leer(A)
	$temp := A * 0,1$
	$A := A$ - $temp$
	escribir(A)
	leer(B)
	$B := B + temp$
	escribir(B)
leer(A)	
$A := A$ - 50	
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	

Planificación 3

- Sean T_1 y T_2 las transacciones definidas anteriormente. La siguiente planificación no es una planificación secuencial, pero es *equivalente* a Planificación 1.

T_1	T_2
leer(A)	
$A := A$ - 50	
escribir(A)	
	leer(A)
	$temp := A * 0,1$
	$A := A$ - $temp$
	escribir(A)
leer(B)	
$B := B$ + 50	
escribir(B)	
	leer(B)
	$B := B + temp$
	escribir(B)

Tanto en la Planificación 1 como en la 3, se preserva la suma A + B.

Planificación 4

- La siguiente planificación concurrente no preserva el valor de la suma *A* + *B*.

T_1	T_2
leer(A)	
$A := A$ - 50	
	leer(A)
	$temp := A * 0,1$
	$A := A$ - $temp$
	escribir(A)
	leer(B)
escribir(A)	
leer(B)	
$B := B + 50$	
escribir(B)	
	$B := B + temp$
	escribir(B)

Secuencialidad

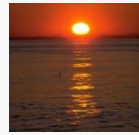
- **Suposición básica** – Cada transacción preserva la consistencia de la base de datos.
- Así, la ejecución secuencial de un conjunto de transacciones preserva la consistencia de la base de datos.
- Una (posiblemente concurrente) planificación es secuenciable si es equivalente a una planificación secuencial. Diferentes formas de equivalencia de planificaciones dan lugar a los conceptos de:
  1. **secuencialidad en cuanto a conflictos**
  2. **secuencialidad en cuanto a vistas**
- Se ignoran la operaciones a excepción de las instrucciones **leer** y **escribir**, y se asume que las transacciones pueden realizar, entre lecturas y escrituras, cálculos arbitrarios sobre datos en memoria intermedia local. Nuestras planificaciones simplificadas constan sólo de instrucciones **leer** y **escribir**.

Secuencialidad en cuanto a conflictos

- Las instrucciones I_i y I_j, de las transacciones T_i y T_j respectivamente, **entran en conflicto** si y sólo si existe algún elemento Q accedido por ambas I_i y I_j, y al menos una de estas instrucciones grabó Q.

1. I_i = **leer**(Q), I_j = **leer**(Q).	I_i y I_j no están en conflicto.
2. I_i = **leer**(Q), I_j = **escribir**(Q).	Están en conflicto
3. I_i = **escribir**(Q), I_j = **leer**(Q).	Están en conflicto
4. I_i =**escribir**(Q), I_j = **escribir**(Q).	Están en conflicto

- Intuitivamente, un conflicto entre I_i y I_j fuerza un (lógico) orden temporal entre ellos.
  - Si I_i y I_j son consecutivos en una planificación y no están en conflicto, sus resultados continuarían siendo los mismos, incluso si se hubieran intercambiado en la planificación.



Secuencialidad en cuanto a conflictos

- Si una planificación *P* se pude transformar en otra *P´* por medio de una serie de intercambios de instrucciones no conflictivas, se dice que *P* y *P´* son **equivalentes en cuanto a conflictos**.
- Se dice que la planificación *P* es **secuenciable en cuanto a conflictos** si es equivalente en cuanto a conflictos a la planificación secuencial *P´*

Secuencialidad en cuanto a conflictos (cont.)

- La planificación 3 se puede transformar en la planificación 6, una planificación secuencial donde T_2 sigue a T_1, mediante conjuntos de intercambios de instrucciones no conflictivas
  - Por lo tanto, la planificación 3 es secuenciable en cuanto a conflictos.

T_1	T_2
leer(A)	
escribir(A)	
	leer(A)
	escribir(A)
leer(B)	
escribir(B)	
	leer(B)
	escribir(B)

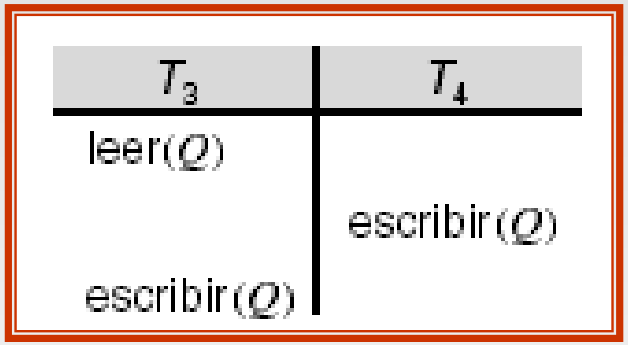
Planificación 3

T_1	T_2
leer(A)	
escribir(A)	
leer(B)	
escribir(B)	
	leer(A)
	escribir(A)
	leer(B)
	escribir(B)

Planificación 6

Secuencialidad en cuanto a conflictos (cont.)

- Ejemplo de una planificación que no es secuenciable en cuanto a conflictos :



- No se pueden intercambiar instrucciones en la planificación anterior para obtener, o la planificación secuencial < T_3, T_4 >, o la planificación secuencial < T_4, T_3 >.

Secuencialidad en cuanto a vistas

- Sean *S* y *S´* dos planificaciones con el mismo conjunto de transacciones. *S* y *S´* son **equivalentes en cuanto a vistas** si se cumplen las tres condiciones siguientes:

  1. Por cada elemento de datos *Q,* si la transacción T_i lee el valor inicial de *Q* en la planificación *S,* entonces la transacción T_i debe, en la planificación *S´*, leer también el valor inicial de *Q.*
  2. Por cada elemento de datos *Q*, si la transacción T_i ejecuta **leer**(*Q)* en la planificación *S* y ese valor fue producido por la transacción T_j (si acaso), entonces la transacción T_i debe leer también, en la planificación *S´*, el valor de *Q* que fue producido por la transacción T_j .
  3. Por cada elemento de daros *Q*, la transacción (si hay) que lleva a cabo la operación final **escribir**(*Q*) en la planificación *S*, debe realizar la operación final **escribir**(*Q*) en la planificación *S´*.

  Como se puede ver, la equivalencia en cuanto a vistas también está totalmente sólo basada en **lecturas** y **escrituras**.

Secuencialidad en cuanto a vistas (cont.)

- Una planificación *S* que es **secuenciable en cuanto a vistas**, es equivalente en cuanto a vistas a una planificación secuencial.
- Cada planificación secuenciable en cuanto a conflictos es también secuenciable en cuanto a vistas.
- A continuación una planificación que es secuenciable en cuanto a vistas pero *no* en cuanto a conflictos.

T_3	T_4	T_6
leer(Q)		
	escribir(Q)	
escribir(Q)		
		escribir(Q)

- ¿Qué planificación secuenciable es equivalente a la anterior?
- Cada planificación secuenciable en cuanto a vistas, que no lo es en cuanto a conflictos, tiene **escrituras a ciegas.**

Otras nociones de secuencialidad

- La siguiente planificación produce los mismos resultados que la planificación secuencial < T_1, T_5 >, aunque no es equivalente a ella en cuanto a conflictos o en cuanto a vistas.

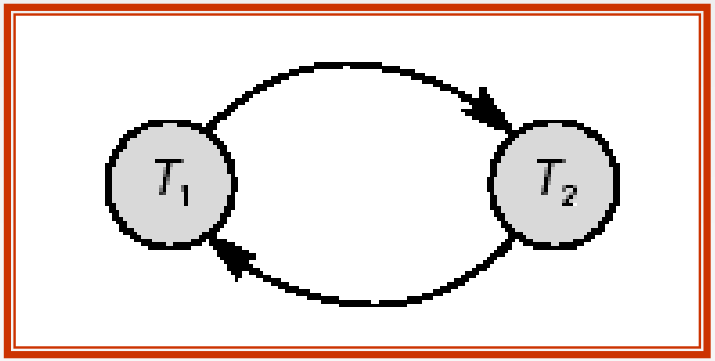
T_1	T_5
leer(A)	
$A := A - 50$	
escribir(A)	
	leer(B)
	$B := B - 10$
	escribir(B)
leer(B)	
$B := B + 50$	
escribir(B)	
	leer(A)
	$A := A + 10$
	escribir(A)

- Determinar tal equivalencia requiere el análisis de operaciones que no sean leer ni escribir.

Prueba de secuencialidad

- Considérese alguna planificación de una conjunto de transacciones T_1, T_2, ..., T_n
- **Grafo de precedencia** – un grafo directo donde los vértices son las transacciones (nombres).
- Se dibuja un arco desde T_i hasta T_j si las dos transacciones están en conflicto, y T_i accedió al elemento de datos sobre el que el conflicto surgió antes.
- Se puede identificar el arco por el elemento al que se accedió.
- **Ejemplo 1**

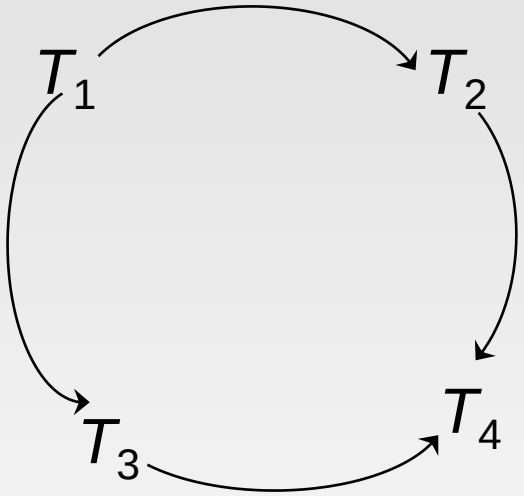
x



y

Ejemplo de planificación (Planificación A) + Grafo de precedencia

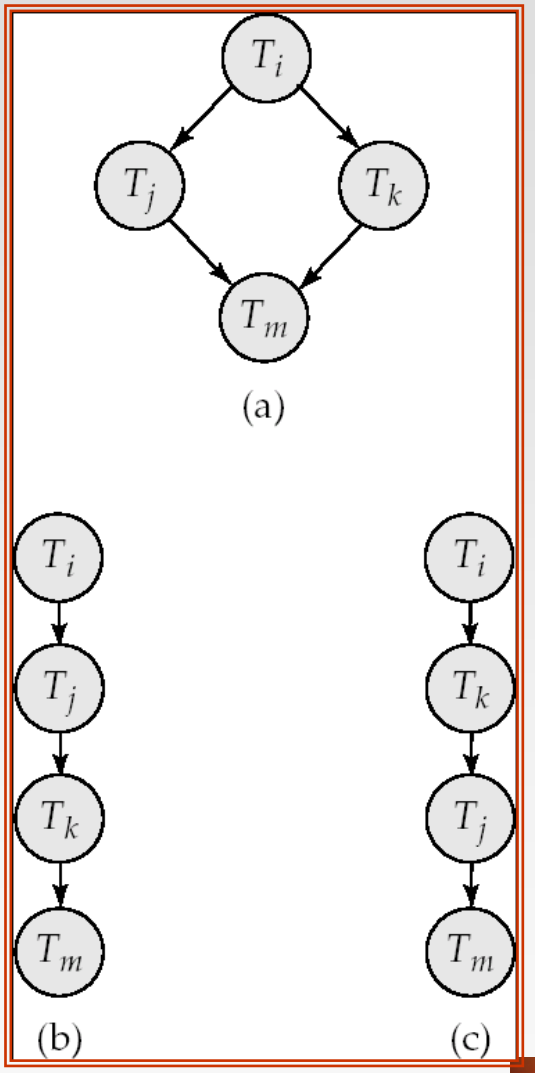
T_1	T_2	T_3	T_4	T_5
	leer(X)			
leer(Y)				
leer(Z)				
				leer(V)
				leer(W)
				leer(W)
	leer(Y)			
	escribir(Y)			
		escribir(Z)		
leer(U)				
			leer(Y)	
			escribir(Y)	
			leer(Z)	
			escribir(Z)	
leer(U)				
escribir(U)				



Prueba para la secuencialidad en cuanto a conflictos

- Una planificación es secuenciable en cuanto a conflictos si y sólo si su grafo de precedencia es acíclico.
- Existen algoritmos de detección de ciclos que toman nota n^2 veces, donde n es el número de vértices del grafo.
  - (Algoritmos mejores toman nota $n + e$ donde e es el número de arcos.)
- Si el grafo de precedencia es acíclico, el orden de secuencialidad se puede obtener por una *ordenación topológica* del grafo.
  - Hay una orden lineal consistente con el orden parcial del grafo.
  - Por ejemplo, un orden de secuencialidad para la planificación A sería

    $T_5 \rightarrow T_1 \rightarrow T_3 \rightarrow T_2 \rightarrow T_4$
    - ¿Existen otros?



Prueba para la secuencialidad en cuanto a vistas

- La prueba del grafo de precedencia, para secuencialidad en cuanto a conflictos, se debe modificar para aplicar una prueba para la secuencialidad en cuanto a vistas.
  - La extensión de la prueba para la secuencialidad en cuanto a vistas tiene un coste exponencial con el tamaño del grafo de precedencia.
- El problema de comprobar si una planificación es secuenciable en cuanto a vistas, falla en la clase de problemas NP-completos.
  - Así la existencia de un algoritmo eficiente es *extremadamente* improbable.
- Sin embargo, todavía se pueden emplear algoritmos prácticos que sólo comprueban algunas **condiciones suficientes** para la secuencialidad en cuanto a vistas.

Recuperabilidad

Necesidad de dirigir el efecto de los fallos en la transacción en transacciones que se ejecutan concurrentemente.

- **Planificación recuperable** — si una transacción T_j lee un elemento de datos previamente escrito por una transacción T_i, la operación comprometer de T_i aparece antes que la operación comprometer de T_j.
- La siguiente planificación (Planificación 11) no es recuperable si T_9 se compromete inmediatamente después de la lectura

T_8	T_9
leer(A)	
escribir(A)	
	leer(A)
leer(B)	

- Si T_8 abortara, T_9 habría leído (y posiblemente mostrado al usuario) un estado inconsistente de la base de datos. Por tanto, la base de datos debe asegurar que las planificaciones son recuperables.

Retroceso en cascada

- **Retroceso en cascada** – un solo fallo en la transacción conduce a un conjunto de retrocesos de la transacción. Considérese la siguiente planificación, donde ninguna de las transacciones se ha comprometido aún (de tal manera que la planificación es recuperable).

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

- Si T_{10} falla, T_{11} y T_{12} también deben ser retrocedidas.
- Puede conducir a la pérdida de una cantidad significativa de trabajo.

Planificaciónes sin cascada

- **Planificaciones sin cascada** – los retrocesos en cascada no pueden tener lugar; para cada par de transacciones T_i y T_j, tales que T_j lee un elemento de datos que ha escrito previamente T_i, la operación comprometer de T_i aparece antes que la operación de lectura de T_j.
- Cada planificación sin cascada es también es recuperable
- Es conveniente restringir las planificaciones a las que son sin cascada

Control de concurrencia

- Una base de datos debe proporcionar un mecanismo que asegure que todas las planificaciones son
  - Serializables en vistas o en conflicto, y
  - Son recuperables y preferiblemente sin cascada
- Una política en que solo se puede ejecutar una transacción a la vez genera planificaciones serie, pero proporciona menor grado de concurrencia
  - ¿Las planificaciones serie son recuperables/sin cascada?
- Comprobar que una planificación es serializable *después* de ejecutarla se un poco tarde.
- **Objetivo** – desarrollar protocolos de control de concurrencia que aseguren la seriabilidad

Control de la concurrencia frente a Pruebas de secuencialidad

- Los protocolos de control de concurrencia permiten planificaciones concurrentes, pero aseguran que las planificaciones son serializables en vistas/en conflicto y son recuperables y sin cascada.
- Los protocolos de control de la concurrencia generalmente no examinan como se crea el grafo de precedencia
  - En cambio un protocolo impondrá una disciplina que evite planificaciones no secuenciables.
  - Tales protocolos se estudiarán en el Capítulo16.
- Los distintos protocolos de control de concurrencia tienen distintas ventajas y desventajas entre la cantidad de concurrencia que permiten y la sobrecarga en que incurren.
- Las pruebas de secuencialidad ayudan a comprender por qué es correcto un protocolo de control de la concurrencia

Niveles débiles de consistencia

- Algunas aplicaciones pueden sobrevivir con niveles débiles de consistencia, permitiendo planificaciones que no son serializables.
  - Ej. Una transacción de sólo lectura que quiere un saldo total aproximado de todas las cuentas
  - Ej. Estadísticas de la base de datos calculadas para la optimización pueden ser aproximadas (¿por qué?)
  - Estas transacciones no hay que serializarlas con respecto a otras.
- Compromiso entre precisión y rendimiento

Niveles de consistencia en SQL-92

- **Secuenciable** — por defecto
- **Lectura repetible** — sólo se pueden leer los registros comprometidos, repetidas lecturas del mismo registro deben devolver el mismo valor. Sin embargo, una transacción puede no ser secuenciable – puede encontrar algunos registros insertados por una transacción, pero no encontrar otros.
- **Con compromiso de lectura** — sólo se pueden leer los registros comprometidos, pero lecturas sucesivas del registro pueden devolver valores diferentes (pero comprometidos).
- **Sin compromiso de lectura**— se pueden leer incluso registros no comprometidos.
- Los niveles de consistencia útiles para reunir información aproximada de la base de datos, p. e., estadísticas del optimizador de consultas

Definición de transacción en SQL

- El lenguaje de manipulación de datos debe incluir una construcción para especificar el conjunto de acciones que comprometen una transacción.
- En SQL, una transacción empieza implícitamente.
- Una transacción en SQL termina por:
  - **Commit work** compromete la transacción actual y empieza una nueva.
  - **Rollback work** origina el aborto de la transacción actual.
- Niveles de consistencia especificados por SQL-92:
  - **Secuenciable** — por defecto
  - **Lectura repetible**
  - **Con compromiso de lectura**
  - **Sin compromiso de lectura**

Fin del capítulo