



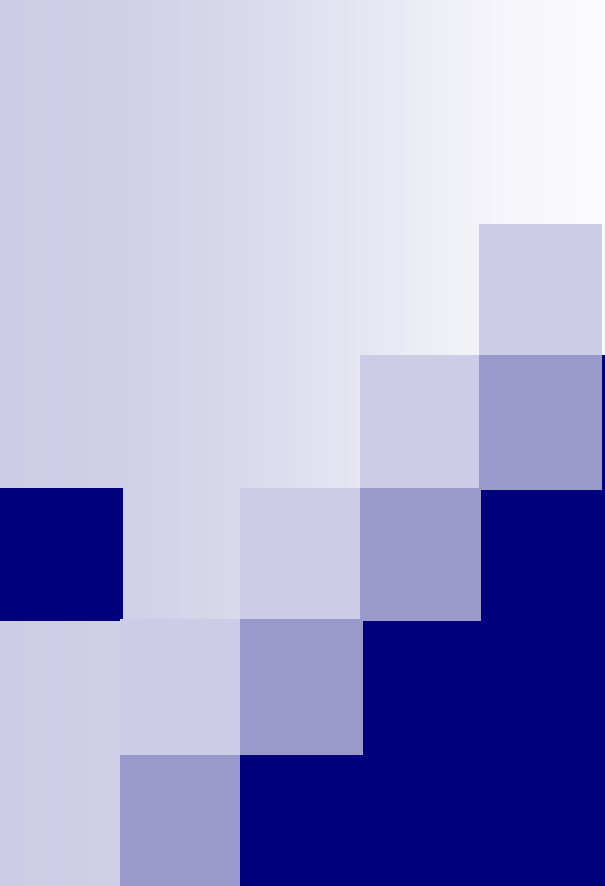
Unidad 3: Transacciones, Concurrencia y Recuperación

Bases de Datos Avanzadas, Sesión 11,12,13 :
Transacciones, Concurrencia y Recuperación

*Iván González Diego
Dept. Ciencias de la Computación
Universidad de Alcalá*

INDICE

- Transacciones.
- Concurrency.
- Recuperación.



Tema 3.1: Transacciones

Tema 5.1: Transacciones

- Concepto de Transacción
- Estados de una Transacción
- Ejecución Concurrente
- Secuencialidad
- Recuperación
- Implementación del Aislamiento
- Definición de Transacción en SQL

Concepto de Transacción

- Una transacción es una unidad de ejecución de programa que accede, y posiblemente actualiza, varios ítems de datos.
- Ej.: Transacción para traspasar 50€ de la cuenta A a la cuenta B:
 - 1.read(A)
 - 2. $A := A - 50$
 - 3.write(A)
 - 4.read(B)
 - 5. $B := B + 50$
 - 6.write(B)
- Dos problemas principales a considerar:
 - ☐ Fallos de varios tipos, tales como fallos de HW y caídas del sistema
 - ☐ Ejecución concurrente de múltiple transacciones

Ej.: Traspaso de Efectivo

- Transacción que traspasa 50 € de la cuenta A a la cuenta B:
 1. read(A)
 2. $A := A - 50$
 3. write(A)
 4. read(B)
 5. $B := B + 50$
 6. write(B)
- Requisito de **Atomicidad**
 - Si la transacción falla después del paso 3 y antes del paso 6, el dinero se habrá “perdido” dando lugar a un estado inconsistente de la Base de Datos
 - Los fallos pueden deberse a SW ó HW
 - El sistema debe asegurar que las actualizaciones de una transacción ejecutada parcialmente no se reflejen en la Base de Datos
- Requisito de **Durabilidad**
 - Si se ha notificado al usuario que la transacción se ha completado (i.e., el traspaso de los 50€ se ha realizado), las actualizaciones en la base de datos deben persistir incluso si hay fallos de HW ó SW

Ej: Traspaso de Efectivo (cont.)

- Transacción que traspasa 50 € de la cuenta A a la cuenta B:

1. read(A)
2. $A := A - 50$
3. write(A)
4. read(B)
5. $B := B + 50$
6. write(B)

- Requisito de **Consistencia**:

- ☐ La suma de A y B no se altera por la ejecución de la transacción
- ☐ En general, el requisito de consistencia incluye:
 - Restricciones de integridad explícitas, como claves primarias y ajenas
 - Restricciones de integridad implícitas
 - ☐ ej: la suma de los saldos de todas las cuentas menos la suma de las cantidades en préstamos debe ser igual al dinero en efectivo
- ☐ Una transacción debe ver una base de datos consistente
- ☐ Durante la ejecución de una transacción la base de datos puede estar temporalmente inconsistente
- ☐ Si la transacción se completa satisfactoriamente la base de datos debe estar consistente
 - Una lógica de transacción errónea puede dar lugar a inconsistencias

Ej: Traspaso de Efectivo (cont.)

■ Requisito de **Aislamiento**

- Si entre los pasos 3 y 6, se permite a otra transacción T2 acceder a la base de datos parcialmente actualizada, verá una base de datos inconsistente (la suma $A + B$ será menor de lo que debería ser)

T1

1. read(A)
2. $A := A - 50$
3. write(A)
4. read(B)
5. $B := B + 50$
6. write(B)

T2

read(A), read(B), print(A+B)

- El aislamiento se puede conseguir trivialmente ejecutando transacciones de forma secuencial: una detrás de otra
- Sin embargo, ejecutar varias transacciones de forma concurrente tiene múltiples ventajas

Propiedades ACID

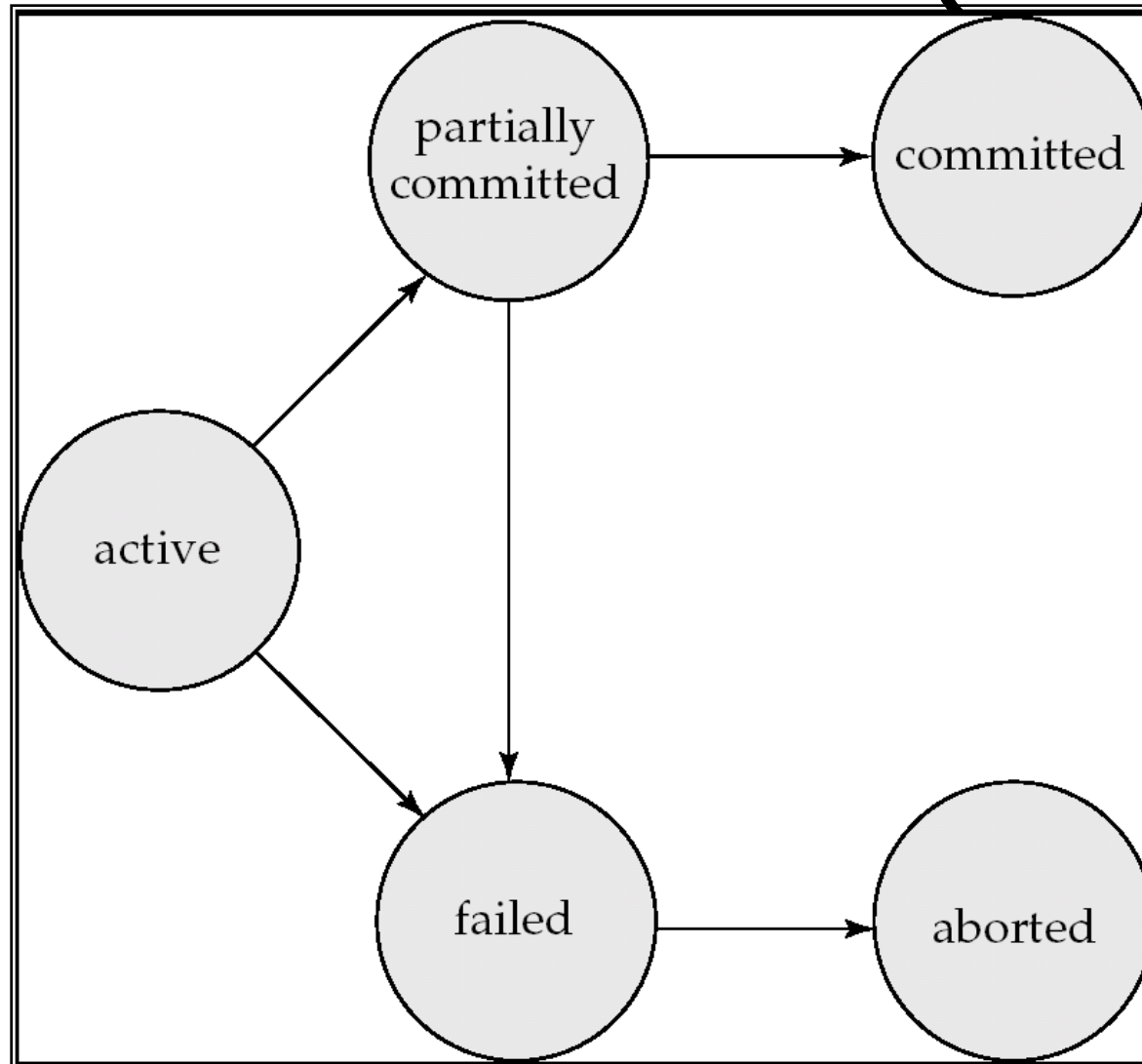
Una transacción es una unidad de ejecución de programa que accede, y posiblemente actualiza, varios ítems de datos. Para mantener la integridad de los datos, un sistema de base de datos debe asegurar:

- **Atomicidad.** Bien todas las operaciones de una transacción están reflejadas en la base de datos, o bien ninguna lo está
- **Consistencia.** La ejecución de una transacción en aislamiento debe mantener la consistencia de la base de datos
- **Isolation** (Aislamiento). Aunque se puedan ejecutar varias transacciones concurrentemente, cada transacción debe ser ignorante de las otras transacciones concurrentes
 - Esto es, para cada par de transacciones T_i y T_j , le parece a T_i que, o bien T_j acabó su ejecución antes de que T_i empezara, o T_j empezó la ejecución tras finalizar T_i
- **Durabilidad.** Si una transacción se completa de forma satisfactoria, los cambios que ésta haya hecho en la base de datos persisten, incluso si hay fallos de sistema

Estados de una Transacción

- **Activa** – el estado inicial; la transacción permanece en este estado mientras se ejecuta
- **Parcialmente comprometida** – tras la ejecución de la última instrucción
- **Fallida** – tras descubrirse que la ejecución normal no puede continuar
- **Abortada** – tras deshacerse (Roll-back) la transacción, y la base de datos haya vuelto a su estado anterior al inicio de la transacción. Dos opciones tras ser abortada:
 - ☐ Reiniciar la transacción
 - se puede hacer sólo si no hay un error lógico interno
 - ☐ Matar la transacción
- **Comprometida** – tras completarse de forma satisfactoria

Estados de una Transacción (cont.)



Ejecución Concurrente

- Se permite ejecutar múltiples transacciones de forma concurrentemente en el sistema
- Ventajas:
 - Incremento de la utilización del procesador y del disco, lo que dan lugar a un mayor rendimiento de transacciones (throughput)
 - Ej. Una transacción puede usar la CPU mientras otra está leyendo o escribiendo en el disco
 - Reducción del tiempo medio de respuesta para las transacciones: las transacciones cortas no necesitan esperar detrás de las transacciones largas
- Esquemas de control de Concurrencia – mecanismos para conseguir el aislamiento
 - Controlar la interacción entre transacciones concurrentes para evitar que destruyan la consistencia de la base de datos
 - En 5.2, tras estudiar las nociones de corrección de ejecución concurrente

Planificación (Schedule)

- **Planificación** (schedule) – secuencias de instrucciones que especifican el orden cronológico en el que se ejecutan las instrucciones de una transacción concurrente
 - Una planificación de un conjunto de transacciones debe incluir todas las instrucciones de esas transacciones
 - Debe mantener el orden en el que las instrucciones aparecen en cada transacción individual
- Una transacción que tiene éxito en completar su ejecución, tendrá una instrucción de *COMMIT* como instrucción final
 - Por defecto, las transacciones suponen que ejecutan una instrucción de *COMMIT* en el último paso
- Una transacción que fracasa en completar su ejecución, tendrá una instrucción de *ABORT* como instrucción final

Planificación 1

- Sea T1 traspasar 50€ de A a B, y T2 traspasar el 10% del saldo de A a B
- Una planificación secuencial (*serial*) en la que T1 es seguida por T2:

T_1	T_2
read(A) $A := A - 50$ write (A) read(B) $B := B + 50$ write(B)	read(A) $temp := A * 0.1$ $A := A - temp$ write(A) read(B) $B := B + temp$ write(B)

Planificación 2

- Una planificación secuencial (*serial*) en la que T2 es seguida por T1:

T_1	T_2
read(A) $A := A - 50$ write(A) read(B) $B := B + 50$ write(B)	read(A) $temp := A * 0.1$ $A := A - temp$ write(A) read(B) $B := B + temp$ write(B)

Planificación 3

- Sean T1 y T2 las transacciones definidas anteriormente
- La siguiente planificación no es una planificación secuencial, pero es equivalente a la Planificación 1

T ₁	T ₂
read(A) $A := A - 50$ write(A)	read(A) $temp := A * 0.1$ $A := A - temp$ write(A)
read(B) $B := B + 50$ write(B)	read(B) $B := B + temp$ write(B)

En las planificaciones 1, 2 y 3, la suma $A + B$ se mantiene

Planificación 4

- La siguiente planificación concurrente no mantiene el valor de $(A + B)$

T_1	T_2
$\text{read}(A)$ $A := A - 50$	$\text{read}(A)$ $\text{temp} := A * 0.1$ $A := A - \text{temp}$ $\text{write}(A)$ $\text{read}(B)$
$\text{write}(A)$ $\text{read}(B)$ $B := B + 50$ $\text{write}(B)$	$B := B + \text{temp}$ $\text{write}(B)$

Secuencialidad

- Supuesto básico – Cada transacción mantiene la consistencia de la base de datos
- Por lo tanto, la ejecución secuencial de un conjunto de transacciones mantiene la consistencia de la base de datos
- Una planificación (posiblemente concurrente) es secuenciable si es equivalente a una planificación secuencial. Diferentes formas de equivalencia de planificación dan lugar a las nociones de:
 1. Secuencialidad en conflictos
 2. Secuencialidad en vistas
- Visión simplificada de las transacciones
 - Ignoraremos todas las operaciones que no sean instrucciones de read y write
 - Supondremos que las transacciones pueden ejecutar cálculos arbitrarios sobre datos en memoria local entre reads y writes.
 - Nuestra planificación simplificada consiste sólo en instrucciones de read y write

Conflicto de Instrucciones

- Las instrucciones li y lj de las transacciones T_i y T_j respectivamente, están en conflicto *si y sólo si* existe algún ítem Q accedido por ambas li and lj , y al menos una de estas instrucciones escribe Q
 - ☐ $li = \text{read}(Q)$, $lj = \text{read}(Q)$ SIN Conflicto
 - ☐ $li = \text{read}(Q)$, $lj = \text{write}(Q)$ Conflicto
 - ☐ $li = \text{write}(Q)$, $lj = \text{read}(Q)$ Conflicto
 - ☐ $li = \text{write}(Q)$, $lj = \text{write}(Q)$ Conflicto
- Intuitivamente, un conflicto entre li y lj fuerza un orden (lógico) temporal entre ellas
 - ☐ Si li y lj están consecutivas en una planificación y no existe conflicto entre ellas, sus resultados serían los mismos, incluso si se intercambiara su orden en la planificación

Secuencialidad en conflictos

- Si una planificación S puede transformarse en otra planificación S' mediante una serie de intercambios de instrucciones que no tienen conflictos, entonces se dice que S y S' son equivalentes en conflictos
- Se dice que una planificación S es secuenciable en conflictos, si es equivalente en conflictos a una planificación secuencial

Secuencialidad en conflictos (cont.)

- La Planificación 3 se puede transformar en la Planificación 6, una planificación secuencial donde T2 sigue a T1, por una serie de intercambios de instrucciones sin conflictos
- Por lo tanto la Planificación 3 es secuenciable en conflictos

T_1	T_2
read(A) write(A)	read(A) write(A)
read(B) write(B)	
	read(B) write(B)

Planificación 3

T_1	T_2
read(A) write(A) read(B) write(B)	read(A) write(A)
	read(B) write(B)

Planificación 6

Secuencialidad en conflictos (cont.)

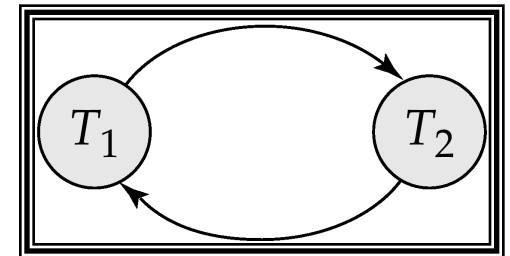
- Ejemplo de planificación que no es secuenciable en conflictos:

T_3	T_4
read(Q)	write(Q)
write(Q)	

- No es posible intercambiar instrucciones en la planificación anterior que generen bien la planificación secuencial $\langle T_3, T_4 \rangle$, o bien la planificación secuencial $\langle T_4, T_3 \rangle$.

Test de Secuencialidad en Conflictos

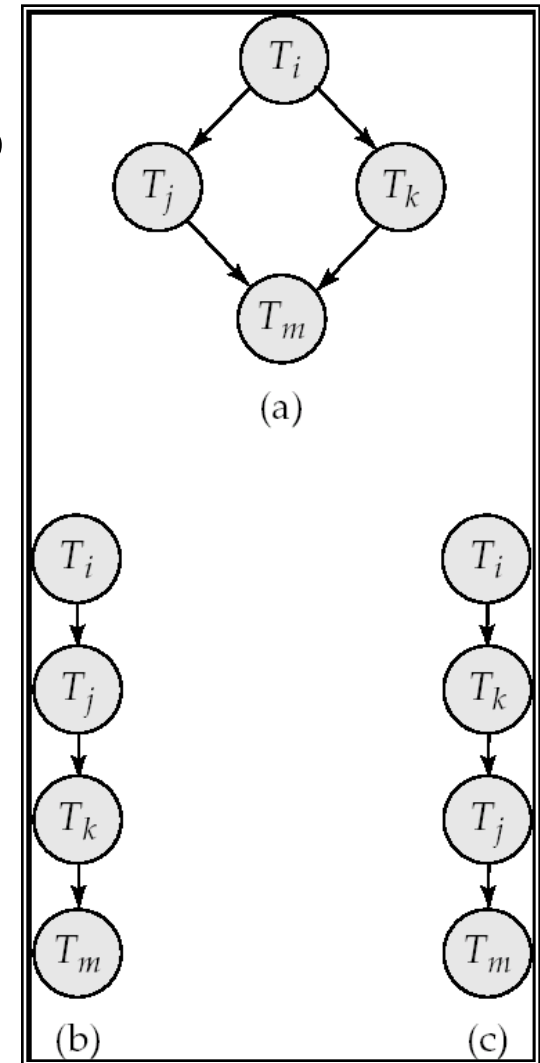
- Planificaciones que se generan son secuenciables.
- Grafo de Precedencia: $G(V,A)$.
 - V es el número de vértices \Rightarrow todas transacciones
 - A un conjunto de arcos: $T_i \rightarrow T_j$, una de las tres condiciones:
 - T_i ejecuta write(Q) antes de que T_j ejecute read(Q)
 - T_i ejecuta read(Q) antes de que T_j ejecute write(Q)
 - T_i ejecuta write(Q) antes de que T_j ejecute write(Q)



Test de Secuencialidad en Conflictos

- Una planificación es secuenciable en conflictos si y sólo si su grafo de precedencias es acíclico
- Existen algoritmos de detección de ciclos con complejidad de orden n^2 en el tiempo, donde n es el número de vértices en el grafo
 - Los mejores algoritmos son de orden $n + e$, 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
 - Es un orden lineal consistente con el orden parcial del grafo
 - Ej.: un orden de secuencialidad para la planificación a) sería

$$T_i \rightarrow T_j \rightarrow T_k \rightarrow T_m$$
 - ¿Hay otros?



Secuencialidad en Vistas

- Sean S y S' dos planificaciones con el mismo conjunto de transacciones. S y S' son equivalentes en vistas si las siguientes tres condiciones se cumplen, para cada ítem de datos Q ,
 1. Si en la planificación S la transacción T_i lee el valor inicial de Q , entonces en la planificación S' también la transacción T_i debe leer el valor inicial de Q
 2. Si en la planificación S , la transacción T_i ejecuta $\text{read}(Q)$, y ese valor estaba producido por la transacción T_j (si existe), entonces en la planificación S' también la transacción T_i debe leer el valor de Q que produjo la misma operación $\text{write}(Q)$ de la transacción T_j
 3. La transacción (si existe) que realiza la operación final $\text{write}(Q)$ en la planificación S , debe también realizar la última operación $\text{write}(Q)$ en la planificación S'
- Como se puede ver, la equivalencia de vistas está también basada en reads y writes únicamente

Secuencialidad en Vistas (cont.)

- Una planificación S es secuenciable en vistas si es equivalente en vistas a una planificación secuencial
- Cada planificación secuenciable en conflictos es también secuenciable en vistas
 - Ej.: Una planificación que es secuenciable en vistas pero no secuenciable en conflictos

T_3	T_4	T_6
read(Q)	write(Q)	
write(Q)		
		write(Q)

- ¿A qué planificación secuencial es equivalente?
- Toda planificación secuenciable en vistas que no sea secuenciable en conflictos tiene escrituras a ciegas (blind writes)

Test de Secuencialidad en Vistas

- El grafo de precedencia para secuencialidad en conflictos no se puede usar directamente como test de secuencialidad en vistas
 - Extensiones para comprobar la secuencialidad en vistas tiene un coste exponencial con el tamaño del grafo de precedencia
- El problema de comprobar si una planificación es secuenciable en vistas pertenece a la clase de problemas NP-completo
 - Por ello, la existencia de un algoritmo eficiente es muy poco probable
 - Sin embargo, se pueden usar algoritmos prácticos que sólo comprueben algunas condiciones suficientes para la secuencialidad en vistas

Otras Nociones de Secuencialidad

- La planificación siguiente produce el mismo resultado que la planificación secuencial $\langle T_1, T_5 \rangle$, a pesar de no ser equivalente en conflictos ni equivalente en vistas a aquella

T_1	T_5
read(A) $A := A - 50$ write(A)	read(B) $B := B - 10$ write(B)
read(B) $B := B + 50$ write(B)	read(A) $A := A + 10$ write(A)

- Determinar tal equivalencia requiere un análisis de las operaciones además de las de read y write.

Planificación Recuperable

Es necesario contemplar el efecto de los fallos de las transacciones en transacciones ejecutándose concurrentemente

- **Planificación Recuperable** — si una transacción T_j lee un ítem de datos previamente escrito por una transacción T_i , entonces la operación de COMMIT de T_i aparece antes de la operación de COMMIT de T_j
- La siguiente planificación (planificación 11) no es recuperable si T_9 ejecuta COMMIT justo después de $\text{read}(A)$

T_8	T_9
Read(A)	
Write(A)	
	Read(A)
COMMIT	
	COMMIT

Recuperable

T_8	T_9
Read(A)	
Write(A)	
	Read(A)
	COMMIT
COMMIT	

No Recuperable

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

Rollback en cascada

- Rollback en cascada – un fallo en una sola transacción da lugar a una serie de vuelta-atrás (rollbacks) de transacción
 - Considerar la siguiente planificación donde ninguna de las transacciones se ha comprometido (por lo que la planificación es recuperable)

T_{10}	T_{11}	T_{12}
read(A) read(B) write(A)	read(A) write(A)	read(A)

Si T_{10} falla, T_{11} y T_{12} deben también volver atrás

- Puede dar lugar a deshacer una gran cantidad de trabajo

Planificación sin Cascada

- **Planificación sin cascada** — No pueden ocurrir Rollbacks en cascada; para cada par de transacciones T_i y T_j tal que T_j lee un ítem de datos previamente escrito por T_i , la operación de COMMIT de T_i aparece antes de la operación de lectura de T_j

T_8	T_9
Write(A)	
	Read(A)
COMMIT	
	COMMIT

Hay Rollback

T_8	T_9
Write(A)	
COMMIT	
	Read(A)
	COMMIT

No hay Rollback

- Cada planificación sin cascada es también recuperable
- Es deseable restringir las planificaciones a aquéllas que son sin cascada

Control de Concurrencia

Implementación de Aislamiento

- Una base de datos debe proporcionar un mecanismo que asegure que todas las posibles planificaciones son
 - ☐ Secuenciables o en conflicto o en vistas, y
 - ☐ Recuperables y preferiblemente sin cascada
- Una política en la que solo se pueda ejecutar una transacción a la vez genera planificaciones secuenciales, pero proporciona un nivel muy pobre de concurrencia
 - ☐ ¿Son las planificaciones secuenciales recuperables/sin cascada?
- Comprobar la secuencialidad de una planificación después de que se haya ejecutado ¡es muy tarde!
- Objetivo – desarrollar protocolos de control de concurrencia que aseguren la secuencialidad

Definición de Transacción en SQL

- El lenguaje de manipulación de datos (DML) debe incluir sentencias para especificar el conjunto de acciones que comprende una transacción
- En SQL, una transacción empieza implícitamente
- Una transacción en SQL acaba con:
 - COMMIT [WORK] compromete la transacción actual y comienza una nueva
 - ROLLBACK [WORK] provoca que la transacción actual aborte
- SQL no especifica que ocurre si se omiten las dos
- En casi cualquier sistema de base de datos, por defecto, cada instrucción de SQL también se compromete de forma implícita si se ejecuta satisfactoriamente
 - El COMMIT implícito se puede desactivar mediante una directiva de la base de datos
 - Ej.: En JDBC, `connection.setAutoCommit(false);`