



Diseño de Bases de Datos

Clase 4

Curso 2015

Prof. Luciano Marrero

Pablo Thomas

Rodolfo Bertone

Agenda

Transacciones

- Propiedades
- Estados

Transacciones monusuarias

- Atomicidad
- Protocolos

Transacciones centralizadas

- Aislamiento
- Consistencia
- Durabilidad

Transacciones

Transacción: colección de operaciones que forman una única unidad lógica de trabajo.

- Propiedades **ACID**
 - Atomicidad: todas las operaciones de la transacción se ejecutan o no lo hacen ninguna de ellas
 - Consistencia: la ejecución aislada de la transacción conserva la consistencia de la BD
 - Aislamiento (isolation): cada transacción ignora el resto de las transacciones que se ejecutan concurrentemente en el sistema, actúa c/u como única.
 - Durabilidad: una transacción terminada con éxito realiza cambios permanentes en la BD, incluso si hay fallos en el sistema

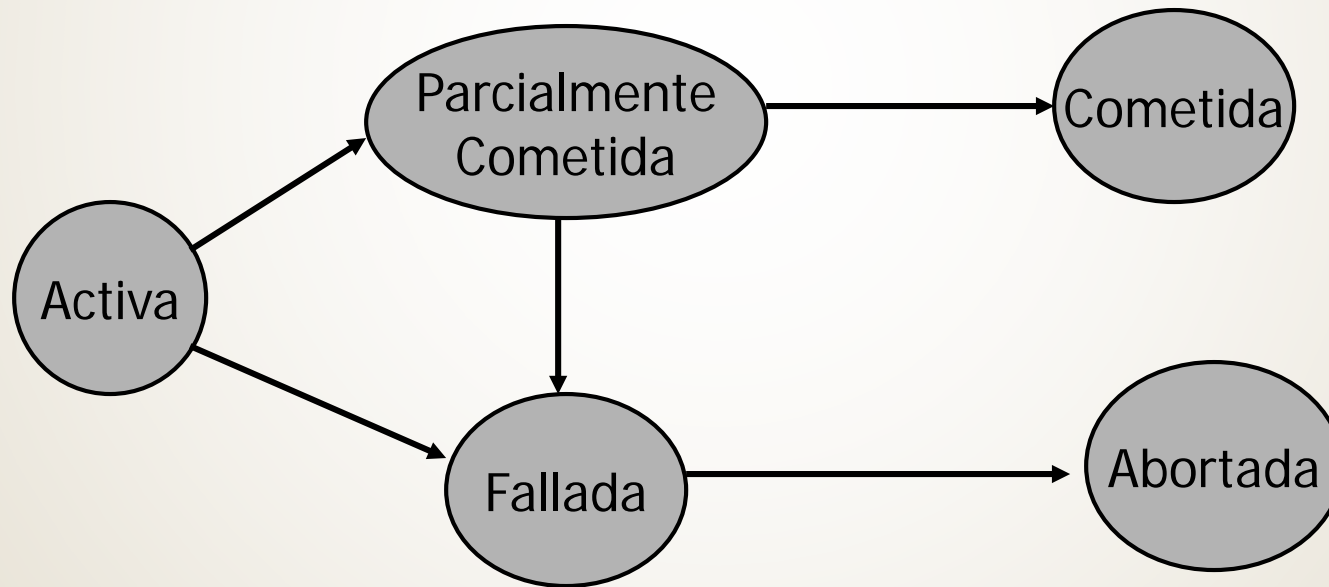
Transacciones

Estados de una transacción

- Activa: estado inicial, estado normal durante la ejecución.
- Parcialmente Cometida: después de ejecutarse la última instrucción
- Fallada: luego de descubrir que no puede seguir la ejecución normal
- Abortada: después de haber retrocedido la transacción y restablecido la BD al estado anterior al comienzo de la transacción.
- Cometida: tras completarse con éxito.

Transacciones

Diagrama de estado de una transacción



Transacciones

Modelo de transacción

- READ (A, a1)
- $a1 := a1 - 100;$
- WRITE(A, a1)
- READ (B, b1)
- $b1 := b1 + 100;$
- WRITE(B, b1)

Diferencia entre READ, WRITE y INPUT, OUTPUT.

Uso de transacciones:

- En sistemas monousuario
- En sistemas concurrentes
- En sistemas distribuidos

Transacciones

Que hacer luego de un fallo?

- Re-ejecutar la transacción fallada → no sirve
- Dejar el estado de la BD como está → no sirve

Problema: modificar la BD sin seguridad que la transacción se va a cometer.

- Solución: indicar las modificaciones

Soluciones

- Registro Historico
- Doble paginación

Registro Histórico

Bitácora

- secuencia de actividades realizadas sobre la BD.
- Contenido de la bitácora
 - <T iniciada>
 - <T, E, Va, Vn>
 - Identificador de la transacción
 - Identificador del elemento de datos
 - Valor anterior
 - Valor nuevo
 - <T Commit>
 - <T Abort>

Registro Historico

Las operaciones sobre la BD deben almacenarse luego de guardar en disco el contenido de la Bitácora

Dos técnicas de bitácora

- Modificación diferida de la BD
- Modificación inmediata de la BD

Registro Histórico

Modificación diferida

- Las operaciones write se aplazan hasta que la transacción esté parcialmente cometida, en ese momento se actualiza la bitácora y la BD

Registro Histórico

Dada la siguiente transacción

- $\langle T_0 \text{ Start} \rangle$
- $\langle T_0, A, 900 \rangle$
- $\langle T_0, B, 2100 \rangle$
- $\langle T_0 \text{ Commit} \rangle$

Recién con T_0 parcialmente cometida, entonces se actualiza la BD.

- No se necesita valor viejo, se modifica la BD al final de la transacción o no se modifica.

Ante un fallo, y luego de recuperarse:

- REDO (T_i), para todo T_i que tenga un Start y un Commit en la Bitácora.
- Si no tiene Commit entonces se ignora, dado que no llegó a hacer algo en la BD.

Registro Histórico

Modificación inmediata:

- La actualización de la BD se realiza mientras la transacción está activa y se va ejecutando.
- Se necesita el valor viejo, pues los cambios se fueron efectuando.
- Ante un fallo, y luego de recuperarse:
 - REDO(T_i), para todo T_i que tenga un Start y un Commit en la Bitácora.
 - UNDO(T_i), para todo T_i que tenga un Start y no un Commit.

Registro Histórico

Transacción:

- Condición de idempotencia.

Buffers de Bitácora

- Grabar en disco c/registro de bitácora insume gran costo de tiempo → se utilizan buffer, como proceder?
- Transacción está parcialmente cometida después de grabar en memoria no volátil el Commit en la Bitácora.
- Un Commit en la bitácora en memoria no volátil, implica que todos los registros anteriores de esa transacción ya están en memoria no volátil.
- **Siempre** graba primero la Bitácora y luego la BD.

Registro Histórico

Puntos de verificación:

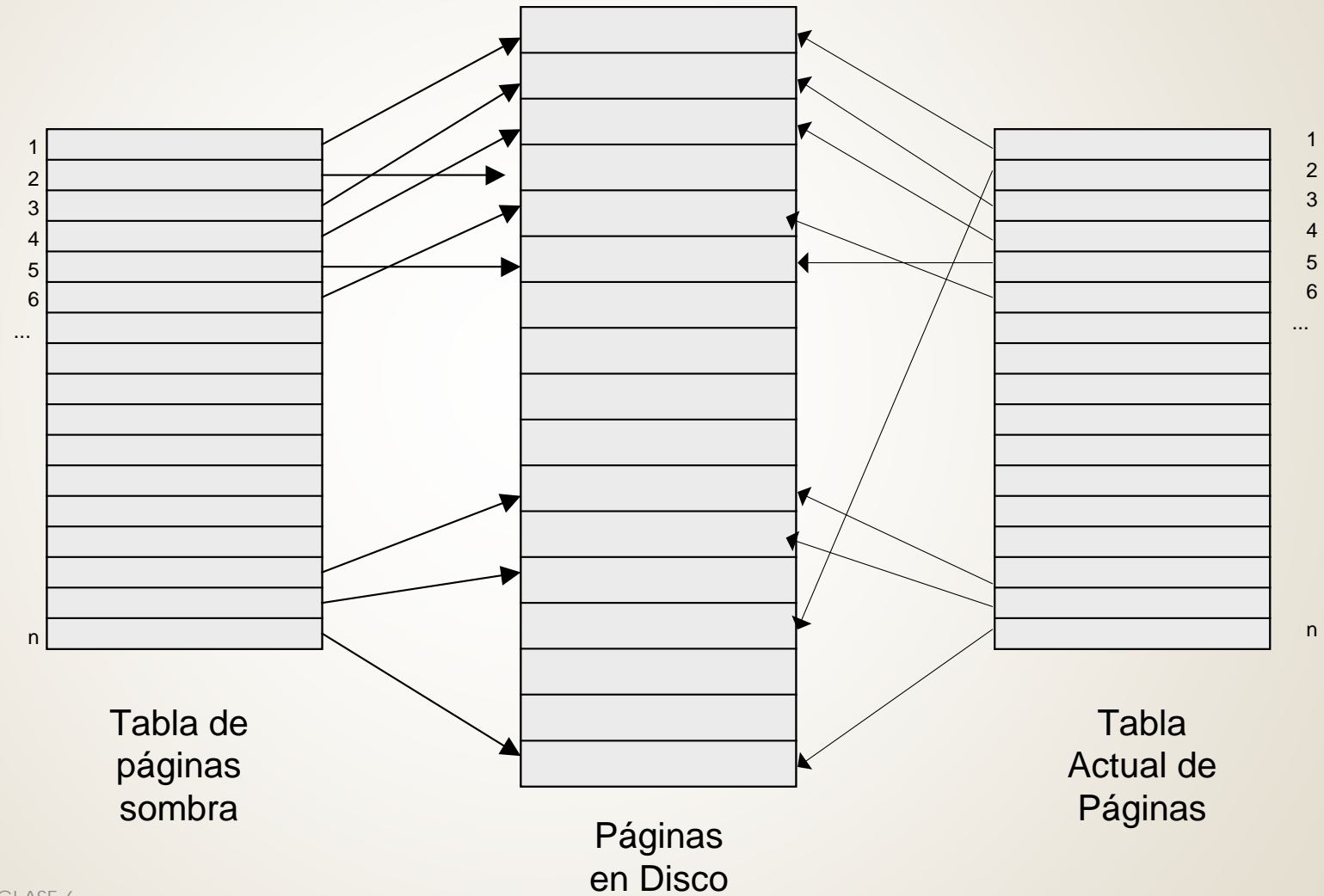
- Ante un fallo, que hacer
 - REDO, UNDO: según el caso
- Revisar la bitácora:
 - Desde el comienzo?: probablemente gran porcentaje esté correcto y terminado.
 - Lleva mucho tiempo.
- Checkpoints (monousario)
 - Se agregan periódicamente indicando desde allí hacia atrás todo OK.
 - Periodicidad?

Doble Paginación

Paginación en la sombra:

- Ventaja: menos accesos a disco
- Desventaja: complicada en un ambiente concurrente/distribuido.
- N páginas equivalente a páginas del SO.
 - Tabla de páginas actual
 - Tabla de páginas sombra

Doble Paginación



Doble Paginación

Ejecución de la operación *escribir*

- Ejecutar **entrada**(X) si página i-ésima no está todavía en memoria principal.
- Si es la primer escritura sobre la página i-ésima, modificar la tabla actual de páginas así:
 - Encontrar una página en el disco no utilizada
 - Indicar que a partir de ahora está ocupada
 - Modificar la tabla actual de página indicando que la i-ésima entrada ahora apunta a la nueva página

Doble Paginación

En caso de fallo y luego de la recuperación

- Copia la tabla de páginas sombra en memoria principal.
- Abort automáticos, se tienen la dirección de la página anterior sin las modificaciones.

Recuperación en caso de Fallo

Ventajas:

- Elimina la sobrecarga de escrituras del log
- Recuperación más rápida (no existe el REDO o UNDO).

Desventajas:

- Sobrecarga en el compromiso: la técnica de paginación es por cada transacción.
- Fragmentación de datos: cambia la ubicación de los datos continuamente.
- Garbage Collector: ante un fallo queda una página que no es mas referenciada.

Entornos Concurrentes

Entorno centralizado

- Varias transacciones ejecutándose simultáneamente compartiendo recursos.
- Deben evitarse los mismos problemas de consistencia de datos
- Transacciones correctas, en ambientes concurrente pueden llevar a fallos

Entornos Concurrentes

Seriabilidad

- Garantiza la consistencia de la BD

```
T0 Read (a)
  a := a - 50
  Write (a)
  Read ( b )
  b := b + 50
  Write ( b )
```

```
T1 Read (a)
  temp := a * 0,1
  a := a - temp
  Write (a)
  Read ( b )
  b := b + temp
  Write ( b )
```

- Resolver T0, T1 o T1, T0 se respeta A+B
- Ahora bien T0 T1 <> T1 T0

Entornos Concurrentes

Planificación: secuencia de ejecución de transacciones

- Involucra todas las instrucciones de las transacciones
- Conservan el orden de ejecución de las mismas
- Un conjunto de m transacciones generan $m!$ planificaciones en serie
- La ejecución concurrente no necesita una planificación en serie.

Entornos concurrentes

```
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)
```

A + B se conserva

```
READ(A)
A := A - 50
    READ(A)
    TEMP := A * 0.1
    A := A - TEMP
    WRITE(A)
    READ(B)
WRITE(A)
READ(B)
B := B + 50
WRITE(B)
    B := B + TEMP
    WRITE(B)
```

A + B no se conserva

Entornos Concurrentes

Conclusiones

- El programa debe conservar la consistencia
- La inconsistencia temporal puede ser causa de inconsistencia en planificaciones en paralelo
- Una planificación concurrente debe equivaler a una planificación en serie
- Solo las instrucciones READ y WRITE son importantes y deben considerarse.

Entornos Concurrentes

Conflicto en planificaciones serializables

- I1, I2 instrucciones de T1 y T2
 - Si operan sobre datos distintos. NO hay conflicto.
 - Si operan sobre el mismo dato
 - I1 = READ(Q) = I2, no importa el orden de ejecución
 - I1 = READ(Q), I2 = WRITE(Q) depende del orden de ejecución (I1 leerá valores distintos)
 - I1 = WRITE(Q), I2 = READ(Q) depende del orden de ejecución (I2 leerá valores distintos)
 - I1 = WRITE(Q) = I2, depende el estado final de la BD
- I1, I2 está en conflicto si actúan sobre el mismo dato y al menos una es un write. Ejemplos.

Entornos Concurrentes

Definiciones

- Una Planificación S se transforma en una S' mediante intercambios de instrucciones no conflictivas, entonces S y S' son *equivalentes en cuanto a conflictos*.
- Esto significa que si
 - S' es consistente, S también lo será
 - S' es inconsistente, S también será inconsistente
- S' es **serializable en conflictos** si existe $S/$ son equivalentes en cuanto a conflictos y S es una planificación serie.

Control de Concurrency

Métodos de control de concurrencia

- Bloqueo
- Basado en hora de entrada

Control de Concurrency

Bloqueo

- Compartido *Lock_c*(dato)(solo lectura)
- Exclusivo *Lock_e*(dato) (lectura/escritura)
- Las transacciones piden lo que necesitan.
- Los bloqueos pueden ser compatibles y existir simultáneamente (compartidos)

Control de Concurrency

Una transacción debe:

- Obtener el dato (si está libre, o compartido y solicita compartido)
- Esperar (otro caso)
- Usar el dato
- Liberarlo.

```
T1  a → b  
    Lock_e(a)  
    Read ( a )  
    a := a - 50  
    Write (a)  
    Unlock ( a )
```

```
    Lock_e(b)  
    Read ( b )  
    b := b + 50  
    Write ( b )  
    Unlock ( b )
```

```
T2  a + b  
    Lock_c(a)  
    Read ( a )  
    Unlock ( a )  
    Lock_c(b)  
    Read ( b )  
    Unlock ( b )
```

$T1 \rightarrow T2$ o $T2 \rightarrow T1$ en serie, no genera problemas

Control de Concurrency

Si se ejecutan en orden verde, azul, celeste
Que pasa con los resultados

Se pueden llevar los bloqueos de T2 ambos al comienzo.

- Puede ocurrir DEADLOCK
- Una de las dos debe retroceder, liberando sus datos.

Control de Concurrency

Conclusiones:

- Si los datos se liberan pronto → se evitan posibles deadlock
- Si los datos se mantienen bloqueados → se evita inconsistencia.

Control de Concurrency

Protocolos de bloqueo

- Dos fases
 - Requiere que las transacciones hagan bloqueos en dos fases:
 - Crecimiento: se obtienen datos
 - Decrecimiento: se liberan los datos
 - Garantiza seriabilidad en conflictos, pero no evita situaciones de deadlock.
- Como se consideran operaciones
 - Fase crecimiento: se piden bloqueos en orden: compartido, exclusivo
 - Fase decrecimiento: se liberan datos o se pasa de exclusivo a compartido.

Control de Concurrency

Protocolo basado en hora de entrada

- El orden de ejecución se determina por adelantado, no depende de quien llega primero
- C/transacción recibe una HDE
 - Hora del servidor
 - Un contador
- Si $HDE(T_i) < HDE(T_j)$, T_i es anterior
- C/Dato
 - Hora en que se ejecutó el último WRITE
 - Hora en que se ejecutó el último READ
 - Las operaciones READ y WRITE que pueden entrar en conflicto se ejecutan y eventualmente fallan por HDE.

Control de Concurrency

Algoritmo de ejecución:

- **Ti Solicita READ(Q)**
 - $HDE(Ti) < HW(Q)$: rechazo (solicita un dato que fue escrito por una transacción posterior)
 - $HDE(Ti) \geq HW(Q)$: ejecuta y se establece $HR(Q) = \text{Max}\{HDE(Ti), HR(Ti)\}$
- **Ti solicita WRITE(Q)**
 - $HDE(Ti) < HR(Q)$: rechazo (Q fue utilizado por otra transaccion anteriormente y suposu que no cambiaba)
 - $HDE(Ti) < HW(Q)$: rechazo (se intenta escribir un valor viejo, obsoleto)
 - $HDE(Ti) > [HW(Q) \text{ y } HR(Q)]$: ejecuta y $HW(Q)$ se establece con $HDE(Ti)$.
- **Si Ti falla, y se rechaza entonces puede recomenzar con una nueva hora de entrada.**

Control de Concurrency

Casos de Concurrency. Granularidad

- A registros caso más normal
- Otros casos
 - BD completa
 - Áreas
 - Tablas

Otras operaciones conflictivas

- Delete(Q) requiere un uso completo del registro
- Insert(Q) el dato permanece bloqueado hasta la operación finalice.

Registro Histórico en entornos concurrentes

Consideraciones del protocolo basado en bitácora

- Existe un único buffer de datos compartidos y uno para la bitácora
- C/transacción tiene un área donde lleva sus datos
- El retroceso de una transacción puede llevar al retroceso de otras transacciones

Retroceso en cascada

- Falla una transacción → pueden llevar a abortar otras
- Puede llevar a deshacer gran cantidad de trabajo.

Registro Histórico en entornos concurrentes

Durabilidad

- Puede ocurrir que falle T_i , y que T_j deba retrocederse, pero que T_j ya terminó. Como actuar?
- Protocolo de bloqueo de dos fases: los bloqueos exclusivos deben conservarse hasta que T_i termine.
- HDE, agrega un bit, para escribir el dato, además de lo analizado, revisar el bit si está en 0 proceder, si está en 1 la transacción anterior no terminó, esperar....

Registro Histórico en entornos concurrentes

Bitácora

- Similar sistemas monousuarios
- Como proceder con checkpoint
 - Colocarlos cuando ninguna transacción esté activa. Puede que no exista el momento.
 - Checkpoint<L> L lista de transacciones activa al momento del checkpoint.
- Ante un fallo
 - UNDO y REDO según el caso.
 - Debemos buscar antes del Checkpoint solo aquellas transacciones que estén en la lista.