Apuntes-SI1-Tema-4.pdf

SalvaGrados

Sistemas Informaticos I

3º Grado en Ingeniería Informática

Escuela Politécnica Superior
Universidad Autónoma de Madrid

TRANSACCIONES

colección de operaciones de lectura y escritura de datos que están relacionadas a nivel lógico (p.ej.: transferencia bancaria) con las siguientes características:

- Atomicidad: Deben ocurrir en su totalidad o no ocurrir en absoluto.
- Si la transacción se ejecuta, los efectos de las operaciones de escritura deben persistir; y si no se completa, la transacción no debe producir ningún efecto.
- Debe implementarse de forma que estos efectos se garanticen incluso si se produce un fallo del sistema (p.ej.. rotura del disco duro, corte en la red, etc.).

Una ejecución (de varias transacciones) es correcta si es **en serie** (una detrás de otra) o **serializable** (equivalente a una ejecución en serie).

En serie	Serializable	No serializable

T1	T2	X	Y
read(X)		25	25
X:= X+100			
write(X)		125	
read(Y)			
Y:= Y+100			
write(Y)			125
	read(X)		
	X:= X*2		
	write(X)	250	
	read(Y)		
	Y:= Y*2		
	write(Y)		250

T1	T2	X	Y
read(X)		25	25
X:= X+100			
write(X)		125	
	read(X)		
	X:= X*2		
	write(X)	250	
read(Y)			
Y:= Y+100			
write(Y)			125
	read(Y)		
	Y:= Y*2		
	write(Y)		250

T1	T2	X	Y
read(X)		25	25
X:= X+100			
write(X)		125	
	read(X)		
	X:= X*2		
	write(X)	250	
	read(Y)		
	Y:= Y*2		
	write(Y)		50
read(Y)			
Y:= Y+100			
write(Y)			150

PROCESOS TRANSACCIONALES

Pasos para una transacción:

1. El cliente solicita la ejecución de un procedimiento remoto especial en el servidor: una transacción.
2. El servidor garantiza la atomicidad de la transacción el sistema nunca queda en un estado inconsistente aunque haya un fallo.
3. El servidor garantiza la correcta ejecución concurrente de las transacciones no hay problemas producidos por accesos múltiples a los datos.

INVARIANTES DE UN SISTEMA

Relaciones que se deben cumplir entre los componentes de un sistema informático.

Ejemplo:

El campo de suma de verificación de una página P debe valer MD5(P)

Para cada elemento en una lista doblemente enlazada se debe verificar que prev(next(x))=x

La tabla EMP1 es una réplica de la tabla EMP

También hay invariantes definidos sobre el modo en que el sistema cambia de estado.

Ejemplo:

Todo cambio en una página debe actualizar su suma de verificación.

Al insertar un elemento en una lista se deben actualizar los punteros de sus elementos anterior y posterior

Cuando se inserta un registro en la tabla EMP se debe insertar también en la tabla EMP1.

PROPIEDADES DE LAS TRANSACCIONES: ACID

ACID: Atomicity, Consistency, Isolation and Durability

- Atomicidad:
  La transacción es una unidad indivisible de trabajo
  Definida con respecto al usuario de la transacción (aplicación que la activa)
- Consistencia:
  Al finalizar su ejecución, el sistema debe quedar en estado estable consistente
  Si no puede realizarla, debe devolver el sistema a su estado inicial (rollback)
- Aislamiento:
  Una transacción no se ve afectada por otras que se ejecuten concurrentemente aunque utilicen los mismos recursos
  Los cambios que introduzca en recursos compartidos no deben ser visibles hasta que la transacción finalice
  Necesario que la transacción "bloquee" los recursos que tiene que actualizar
- Durabilidad:
  Sus efectos son permanentes una vez que ha finalizado correctamente (commit)

WUOLAH + BBVA

1/6	BBVA está adherido al Fondo de Garantía de Depósitos de Entidades de Crédito de España. La cantidad máxima garantizada es de 100.000 euros por la totalidad de los depósitos constituidos en BBVA por persona.
Este número es indicativo del riesgo del producto, siendo 1/6 indicativo de menor riesgo y 6/6 de mayor riesgo.	

Te regalamos

15€

1

Abre tu Cuenta Online **sin comisiones ni condiciones**

2

Haz una compra igual o superior a 15€ **con tu nueva tarjeta**

3

BBVA te devuelve un **máximo de 15€**

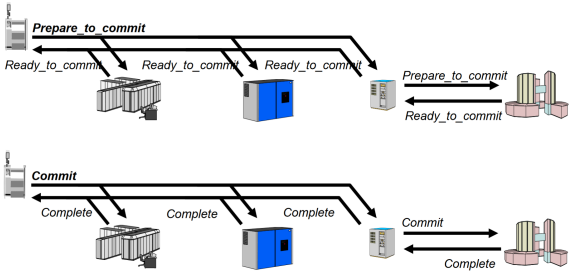
TIPOS DE TRANSACCIONES

Transacciones Planas (flat transactions):
Todo el proceso realizado en su interior se encuentra al mismo nivel de jerarquía (no hay llamadas a otras transacciones)
Comienzan con un comando tipo begin_transaction
Finalizan:

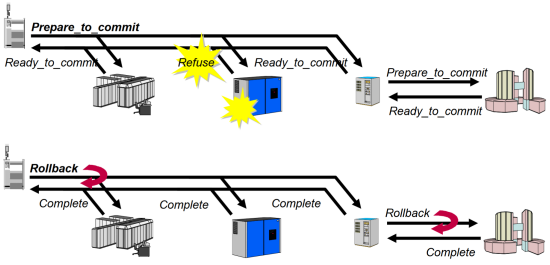
- Si el proceso fue correcto: comando tipo commit_transaction. Todo lo realizado por la transacción se valida y es accesible al exterior.
- Si ha fallado el proceso: comando tipo rollback_transaction. Todos los cambios que hubiera realizado la transacción se deshacen.

Transacciones Planas Distribuidas:
Desde el punto de vista de la aplicación que la utiliza es una transacción plana, pero en realidad trabajan con recursos ubicados en diferentes sistemas.
El monitor de proceso transaccional que la controla debe garantizar que la transacción continúa siendo ACID:

- Necesario garantizar la destrucción de las operaciones en todos los sistemas afectados en caso de que algo falle
- Requiere un nuevo proceso de validación de la transacción: **Two-Phase Commit.** Consiste en lanzar primero un Prepare desde el cliente donde los diferentes servidores devuelven un Ready. La segunda fase es lanzar el Commit desde el servidor, para que finalmente los servidores devuelvan el Complete (Ver imagen).



Si en vez de un Ready un servidor respondiese Refuse, se haría Rollback (Volver a la situación original en todos los servidores).



Transacciones Anidadas:
Llamada a una transacción desde otra (modelo subrutinas). Un commit de una transacción hace sus cambios visibles a todas las transacciones precedentes en la jerarquía de llamadas
Un rollback de una transacción realiza un rollback de todas las subtransacciones que haya ejecutado, aunque estas ya hayan realizado un commit.

Transacciones con Savepoints:
Para evitar el efecto "todo-o-nada", también se ha propuesto utilizar savepoints. Registran estados consistentes dentro de la transacción. Cuando sea necesario un rollback, en principio no es necesario deshacer toda la transacción, sino que se vuelve al último savepoint.
Los efectos de la transacción sólo serán permanentes y visibles "desde fuera" con el commit de la transacción.

Los Savepoints y las Transacciones Anidadas no resuelven el problema de la caída de un servidor en plena transacción.

AISLAMIENTO. CONTROL DE LA CONCURRENCIA
Aislamiento (isolation): Propiedad de los sistemas de proceso transaccional por la cual una transacción no se ve afectada por otras que se ejecuten concurrentemente aunque utilicen los mismos recursos
Leyes de la concurrencia:

- Primera ley: La ejecución concurrente no debe causar que los programas de aplicación funcionen incorrectamente.
  Consecuencias:
  - Todo programa debe ver un estado consistente de los recursos al comienzo de su ejecución.
  - Toda modificación del estado consistente de los recursos que vea un programa debe ser motivada por él mismo.
- Segunda ley: La ejecución concurrente no debe tener menor rendimiento o tiempos de respuesta mucho mayores que la ejecución en serie.
  Consecuencia:
  - Los algoritmos de control de la concurrencia deben ser sencillos y eficientes.

DEFINICIONES SOBRE EL AISLAMIENTO

Historia de un conjunto de transacciones: Secuencia que preserva la mezcla de acciones que tiene lugar al ejecutarse un conjunto de transacciones. (Con el ejemplo de abajo se entiende fácil).

$$H = \left\langle \langle t, a, o \rangle_i \,|\, i = 1,\ldots,n \right\rangle$$

Versiones de objetos:

- Cada variación de un objeto del sistema se puede ver como la creación de una nueva versión del objeto: <o, 1>, <o, 2>, <o, 3>...
- Las versiones de los objetos se crean por acciones de escritura en las transacciones.
- La versión de un objeto "o" en un paso k de una historia se denota por V(o,k), y representa el número de escrituras realizadas sobre el objeto en la historia H hasta llegar al paso k.

$$V(o,k) = \left| \left\{ \langle t, a, o \rangle_j \in H \,|\, (j < k) \wedge (a_j = WRITE) \wedge (o_j = o) \right\} \right|$$

Ejemplo 1

Acciones:

```
T1 READ A
T2 WRITE C
T3 WRITE C
T2 READ B
```

Historia:

```
H = < <T1, READ, A>, <T2, WRITE, C>, <T3, WRITE, C>, <T2, READ, B> >
```

Evolución de las versiones de los objetos:

```
V(A,1)=0 V(A,2)=0 V(A,3)=0 V(A,4)=0
V(B,1)=0 V(B,2)=0 V(B,3)=0 V(B,4)=0
V(C,1)=0 V(C,2)=0 V(C,3)=1 V(C,4)=2
```

Ejemplo 2: Historias en serie y serializable (Ejemplo del inicio del tema)

En serie:

```
H1 = < <T1, READ, A>,<T1, WRITE, A>,<T1, READ, B>,
<T1, WRITE,B>, <T2, READ, A>,<T2, WRITE, A>,<T2, READ, B>,
<T2, WRITE, B> >
```

Serializable:

```
 H2 = < <T1, READ, A>,<T1, WRITE, A>, <T2, READ, A>,
<T2,WRITE, A>, <T1, READ, B>,<T1, WRITE, B>,<T2, READ, B>,
<T2, WRITE, B> >
```

No serializable:

```
H3 = < <T1, READ, A>,<T1, WRITE, A>, <T2, READ,A>,
<T2, WRITE, A>,<T2, READ, B>,<T2, WRITE, B>, <T1, READ, B>,
<T1,WRITE, B> >
```

TIPOS DE VIOLACIONES DE AISLAMIENTO

Actualización perdida (lost update)
La escritura de una transacción es ignorada por otra, que realiza una nueva escritura basada en la versión previa del objeto.

```
T1 READ <o,1>
T2 READ <o,1>
T1 WRITE <o,2>
T2 WRITE <o,3>
```

Lectura sucia (dirty read)
Una transacción lee un objeto escrito por otra en un estado intermedio.

```
T2 READ <o,1>
T2 WRITE <o,2>
T1 READ <o,2>
T2 WRITE <o,3>
```

Lectura no repetible (unrepeatable read)
Una transacción lee un objeto dos veces con valores distintos, por haber una actualización intermedia realizada por otra transacción.

```
T1 READ <o,1>
T2 WRITE <o,2>
T1 READ <o,2>
```

Lectura Fantasma
Parecido a lectura no repetible pero con un insert.

```
T1 READ <o,1>
T2 INSERT <o,2>
T1 READ <o,2>
```

BLOQUEO (LOCK)
Acción de una transacción por la cual se señala el uso de un objeto.
Dos tipos de bloqueos:

- Bloqueo compartido (Shared Lock): No se requiere el uso exclusivo del objeto.
- Bloqueo exclusivo (Exclusive Lock): Se requiere el uso exclusivo del objeto.

Compatibilidad de bloqueos de un mismo objeto entre transacciones:

Compatibilidad		Modo de bloqueo existente	
		Compartido	**Exclusivo**
Modo bloqueo solicitado	**Compartido**	Compatible	Prohibido
	Exclusivo	Prohibido	Prohibido

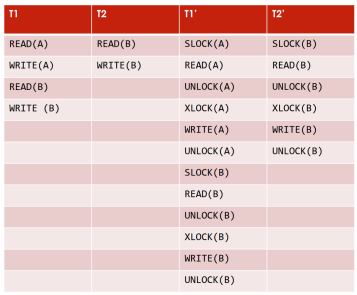
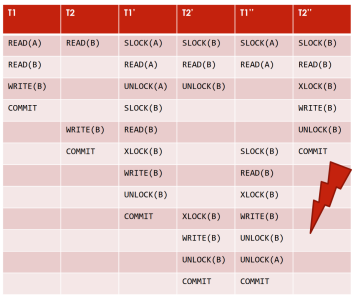
ACCIONES DENTRO DE UNA TRANSACCIÓN

Acciones sobre objetos	Acciones genéricas:
READ: Lectura de un objeto. **WRITE**: Escritura de un objeto. **XLOCK** (eXclusive LOCK): Solicitud uso exclusivo de un objeto. **SLOCK** (Shared LOCK): Solicitud de uso compartido de un objeto. **UNLOCK**: Liberación de una solicitud de uso.	**BEGIN** **COMMIT**: Equivale a la liberación de todos los bloqueos realizados por la transacción. **ROLLBACK**: Equivale a deshacer todas las escrituras realizadas por la transacción y liberar todos sus bloqueos.

TRANSACCIONES BIEN FORMADAS
Todas sus lecturas y escrituras están cubiertas por bloqueos (precedidas por un lock del tipo adecuado y no realizado un unlock)
READ: cubierta (al menos) por SLOCK
WRITE: cubierta por XLOCK

TRANSACCIONES EN DOS FASES
Todos los bloqueos preceden a todos los desbloqueos
Fase de crecimiento: adquiere los bloqueos
Fase de contracción: libera los bloqueos

Estos dos tipos de transacciones son compatibles y sus propiedades pueden combinarse para aumentar su grado de aislamiento.

Bien Formada	Dos Pasos	Aislamiento Completo
		

AISLAMIENTO COMPLETO

Para garantizar el aislamiento de cualquiera de las combinaciones posibles de un conjunto de transacciones se deben programar atendiendo a los siguientes criterios:

- Escribir siempre transacciones bien formadas. Proteger todas las acciones con bloqueos.
- Establecer bloqueos exclusivos en los objetos que se vayan a actualizar.
- Escribir transacciones en dos fases: No comenzar a liberar los bloqueos hasta que no se necesite realizar más (bloqueos)
- Mantener los bloqueos exclusivos hasta que se realice el Commit o el Rollback.

El aislamiento completo no evita las lecturas fantasma asociadas a inserciones y borrado de objetos que deberían estar bloqueados.

GRADOS DE AISLAMIENTO

El aislamiento completo es costoso

Muchos fabricantes permiten activar selectivamente el grado de aislamiento deseado para el sistema:

- 0º (grado 0) caos. Las transacciones no sobreescriben datos sucios de transacciones de nivel 1º o superior
- 1º (grado 1) lecturas no comprometidas. No hay pérdida de actualizaciones en el ámbito de la transacción (hasta el COMMIT)
- 2º (grado 2) lecturas comprometidas. No hay pérdida de actualizaciones en el ámbito de la transacción ni lectura de datos sucios
- 3º (grado 3) lectura repetible. No hay pérdida de actualizaciones ni lectura de datos sucios y hay lecturas repetitivas. Es el aislamiento completo garantizado.
- 4º (grado 4) Serializable. Imita el comportamiento en serie. No hay razón para usarlo.

Menor grado de aislamiento facilita la concurrencia y evita esperas en las transacciones, pero puede producir violaciones de aislamiento.

Aspecto	Grado 0	Grado 1	Grado 2	Grado 3
Nombre	Caos	*Read Uncommited* Consulta *(Browse)*	*Read Commited*	*Repeatable Read*
Dependencias consideradas	Ninguna	WRITE → WRITE	1° y WRITE → READ	2° y READ → WRITE
Protección proporcionada	Permite los niveles superiores	0° y no se pierden actualizaciones	1° y no hay lecturas *sucias*	2° y hay lecturas repetibles
Datos validados *(committed)*	Escrituras visibles inmendiatamente	Escrituras visibles al fin de la transacción	Igual que 1°	Igual que 1°
Datos sucios	No se sobreescriben datos *sucios* ajenos	Nadie sobreescribe datos *sucios*	1° y no se leen datos *sucios*	2° y no se *ensucian* datos ya leidos
Protocolo de bloqueo	Bloqueos exclusivos cortos en escrituras	Bloqueos exclusivos largos en escrituras	1° y bloqueos cortos compartidos lectura	1° y bloqueos largos compartidos lectura
Estructura de la transacción	Bien formadas (Wrt)	Bien formadas (Wrt) Dos fases (Wrt)	Bien formadas. Dos fases (Wrt)	Bien formadas Dos fases.
Concurrencia	Muy alta. Casi no hay esperas	Alta: Sólo espera bloqueos escritura	Media: 1° y bloqueo en algunas lecturas	Baja: bloqueo se mantiene hasta EOT
Sobrecarga *(overhead)*	Mínima.	Pequeña	Media	Media
Rollback	No existe	Funciona	Igual que 1°	Igual que 1°
Recuperación del	Peligroso. Pérdidas y	Aplicar log en orden 1°	Igual que 1°	Aplicar log en orden <<<

GRADO DE AISLAMIENTO ESTABILIDAD DE CURSORES

El grado de aislamiento "estabilidad de cursores" (cursor stability, no disponible en todos los SGBD) mejora el grado 2 para resolver el problema anterior.

Se utiliza en sentencias con cursores.

Mantiene el bloqueo compartido sobre el registro al que accede el cursor hasta que se pase al siguiente registro.

Si se realiza una actualización, el bloqueo se convierte en exclusivo, y, por tanto, se mantiene.

Si se lee el siguiente registro, el bloqueo se libera.

BLOQUEOS FOR UPDATE

Bloqueo para cursores declarados para realizar actualizaciones sobre los registros encontrados:

Si se realiza una actualización, el bloqueo se convierte en exclusivo, y, por tanto, se mantiene hasta el final de la transacción

Si no se actualiza y se pasa a leer el siguiente registro:

- El bloqueo se mantiene hasta el final de la transacción si ésta es de grado 3.
- El bloqueo se libera si la transacción es de grado 2

INTERBLOQUEO (DEADLOCK)

• Situación de bloqueo recíproco de recursos, que producen espera ilimitada:

```
T1 XLOCK A T2 XLOCK B
T1 XLOCK B T2 XLOCK A
```

Solución más simple: nunca parar. En caso de que se deniegue un bloqueo, ejecutar rollback. Intentar de nuevo la transacción.

- Puede producir situaciones de livelock.

Detección por timeout: cancelar la transacción tras un tiempo de espera.

- Última solución para liberar algunos interbloqueos
- La espera en un bloqueo es una situación muy rara. Un interbloqueo es aún mucho más raro
- Los mecanismos de detección deben ser sencillos, baratos y no deben sobrecargar la ejecución normal

T1	T2	T1'	T2'
READ(A)	READ(B)	SLOCK(A)	SLOCK(B)
WRITE(A)	WRITE(B)	READ(A)	READ(B)
READ(B)	READ(A)	XLOCK(A)	XLOCK(B)
WRITE(B)	WRITE(A)	WRITE(A)	WRITE(B)
		SLOCK(B)	SLOCK(A)

DEADLOCK!!

Grafo de esperas: Transacciones del sistema en los nodos. Existe un arco entre las transacciones T y T' si:

T está esperando a un recurso bloqueado por T'.

T y T' están ambas en espera de un recurso, T está detrás de T' en la lista de espera y sus peticiones son incompatibles.

Si existe un ciclo en el grafo de esperas, es un interbloqueo.

```
T1    XLOCK   A
T2    XLOCK   B
T3    XLOCK   C
T4    XLOCK   D
T2    XLOCK   C
T3    XLOCK   D
T4    XLOCK   B
T1    XLOCK   D
```